ewoajsmjo

April 26, 2025

1 Importing Libraries

[ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import re
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder,StandardScaler
from category_encoders import TargetEncoder
from sklearn.model_selection import train_test_split,learning_curve
from sklearn.linear_model import LogisticRegression
import mlflow
import mlflow.sklearn
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score,
    fbeta_score, roc_curve, auc, precision_recall_curve,
    classification_report, confusion_matrix, ConfusionMatrixDisplay
)
from imblearn.over_sampling import SMOTE
import statsmodels.api as sm
import warnings
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

[ ]:
```
# Changing the pandas option to display all columns of table
pd.set_option('display.max_columns',None)
```

2 Importing Dataset

[ ]:
```
loan_data = pd.read_csv('logistic_regression.csv')
```

[ ]:
```
loan_data.head()
```

[ ]:
```
   loan_amnt        term  int_rate  installment grade sub_grade  \
0    10000.0   36 months     11.44       329.48     B        B4
1     8000.0   36 months     11.99       265.68     B        B5
```

```
2    15600.0   36 months     10.49       506.97     B        B3
3     7200.0   36 months      6.49       220.65     A        A2
4    24375.0   60 months     17.27       609.33     C        C5

                emp_title emp_length home_ownership  annual_inc  \
0               Marketing  10+ years           RENT    117000.0
1         Credit analyst     4 years       MORTGAGE     65000.0
2            Statistician   < 1 year           RENT     43057.0
3         Client Advocate    6 years           RENT     54000.0
4  Destiny Management Inc.    9 years       MORTGAGE     55000.0

  verification_status   issue_d  loan_status             purpose  \
0       Not Verified  Jan-2015   Fully Paid            vacation
1       Not Verified  Jan-2015   Fully Paid  debt_consolidation
2    Source Verified  Jan-2015   Fully Paid         credit_card
3       Not Verified  Nov-2014   Fully Paid         credit_card
4           Verified  Apr-2013  Charged Off         credit_card

                   title    dti earliest_cr_line  open_acc  pub_rec  \
0                Vacation  26.24         Jun-1990      16.0      0.0
1      Debt consolidation  22.05         Jul-2004      17.0      0.0
2  Credit card refinancing  12.79         Aug-2007      13.0      0.0
3  Credit card refinancing   2.60         Sep-2006       6.0      0.0
4    Credit Card Refinance  33.95         Mar-1999      13.0      0.0

   revol_bal  revol_util  total_acc initial_list_status application_type  \
0    36369.0        41.8       25.0                  w       INDIVIDUAL
1    20131.0        53.3       27.0                  f       INDIVIDUAL
2    11987.0        92.2       26.0                  f       INDIVIDUAL
3     5472.0        21.5       13.0                  f       INDIVIDUAL
4    24584.0        69.8       43.0                  f       INDIVIDUAL

   mort_acc  pub_rec_bankruptcies  \
0       0.0                   0.0
1       3.0                   0.0
2       0.0                   0.0
3       0.0                   0.0
4       1.0                   0.0

                                          address
0     0174 Michelle Gateway\r\nMendozaberg, OK 22690
1  1076 Carney Fort Apt. 347\r\nLoganmouth, SD 05113
2  87025 Mark Dale Apt. 269\r\nNew Sabrina, WV 05113
3            823 Reid Ford\r\nDelacruzside, MA 00813
4             679 Luna Roads\r\nGreggshire, VA 11650
```

3 Description regaarding the dataset columns

Column Name	Description
loan_amnt	The listed amount of the loan applied 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. **Formulae: EMI = installment = (loan_amnt * (int_rate/100) * (1+(int_rate/100))$^{\text{term})/((1+(\text{int\_rate}/100))}$(term-1))**
grade	LoanTap assigned loan grade.(**Loan grading is the process of assigning a quality score to a loan application to identify a risk of default. This score is based on the borrower's credit history, quality of the collateral, and likelihood of repayment.Generally A grade means Lower interest, Lower loan losses, Lower expected returns. G grade means Higher interest, Higher loan losses, Higher expected returns**)
sub_grade	LoanTap assigned loan subgrade(**each loan grade is again subdivided into subgrades. Generally 1 means lowest interest rate in that loan grade. 5 means highest interest rate in that loan grade**)
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**

Column Name	Description
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. **total debt(excluding mortgage and loan tap loan)/monthly income**
earliest_cr_line	The month the borrower's earliest reported credit line was opened (**credit line means a debt account**)
open_acc	The number of open credit lines in the borrower's credit file. **implies number of debt accounts at present**
pub_rec	Number of derogatory public records **implies number of negative records publicly**
revol_bal	Total credit revolving balance **implies present total credit amount yet to be paid by the borrower**
revol_util	Revolving line utilization rate, or the amount of credit the borrower is using relative to all available revolving credit. **implies total debt/total available credit**
total_acc	The total number of credit lines currently in the borrower's credit file **implies total number of debt accounts opened in borrower life. Some of them may be closed/cleared by now.**
initial_list_status	The initial listing status of the loan. Possible values are – W, F. **W = Whole loan amount funded by single investor, F = Fractional Loan invest implies Number of investors funded the loan amount**
application_type	Indicates whether the loan is an individual application or a joint application with two co-borrowers
mort_acc	Number of mortgage accounts **implies Number of accounts related to home loan or real estate related loan**
pub_rec_bankruptcies	Number of public record bankruptcies **implies number of times borrower has filed bankruptcies in court**
Address	Address of the individual

4 Exploratory Data Analysis

4.1 Info regarding the dataset

[ ]:
```
loan_data.shape
```

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

[ ]:
```
loan_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
```

4.2 Descriptive Statistics

[ ]:
```
loan_data.describe().T
```

[ ]:

	count	mean	std	min	25%	\
loan_amnt	396030.0	14113.888089	8357.441341	500.00	8000.00	
int_rate	396030.0	13.639400	4.472157	5.32	10.49	
installment	396030.0	431.849698	250.727790	16.08	250.33	
annual_inc	396030.0	74203.175798	61637.621158	0.00	45000.00	
dti	396030.0	17.379514	18.019092	0.00	11.28	
open_acc	396030.0	11.311153	5.137649	0.00	8.00	
pub_rec	396030.0	0.178191	0.530671	0.00	0.00	
revol_bal	396030.0	15844.539853	20591.836109	0.00	6025.00	
revol_util	395754.0	53.791749	24.452193	0.00	35.80	
total_acc	396030.0	25.414744	11.886991	2.00	17.00	
mort_acc	358235.0	1.813991	2.147930	0.00	0.00	
pub_rec_bankruptcies	395495.0	0.121648	0.356174	0.00	0.00	

	50%	75%	max
loan_amnt	12000.00	20000.00	40000.00
int_rate	13.33	16.49	30.99
installment	375.43	567.30	1533.81
annual_inc	64000.00	90000.00	8706582.00
dti	16.91	22.98	9999.00
open_acc	10.00	14.00	90.00
pub_rec	0.00	0.00	86.00
revol_bal	11181.00	19620.00	1743266.00
revol_util	54.80	72.90	892.30
total_acc	24.00	32.00	151.00
mort_acc	1.00	3.00	34.00
pub_rec_bankruptcies	0.00	0.00	8.00

[ ]:
```
loan_data.describe(include = 'object').T
```

[ ]:

	count	unique	top	freq
term	396030	2	36 months	302005
grade	396030	7	B	116018
sub_grade	396030	35	B3	26655
emp_title	373103	173105	Teacher	4389
emp_length	377729	11	10+ years	126041
home_ownership	396030	6	MORTGAGE	198348
verification_status	396030	3	Verified	139563
issue_d	396030	115	Oct-2014	14846
loan_status	396030	2	Fully Paid	318357
purpose	396030	14	debt_consolidation	234507
title	394274	48816	Debt consolidation	152472
earliest_cr_line	396030	684	Oct-2000	3017

```
initial_list_status  396030       2                          f  238066
application_type     396030       3                 INDIVIDUAL  395319
address              396030  393700  USS Smith\r\nFPO AP 70466       8
```

[ ]:
```
loan_data.duplicated().sum()
```

[ ]:
```
np.int64(0)
```

4.3 Missing value Detection

[ ]:
```
# Filtering the columns which have zero null values and sorting in descending␣
 ↪order
def missing_loan_data(loan_data):
    total_missing_loan_data = loan_data.isna().sum().sort_values(ascending =␣
 ↪False)
    percentage_missing_loan_data = ((loan_data.isna().sum()/
 ↪len(loan_data)*100)).sort_values(ascending = False)
    missingloan_data = pd.concat([total_missing_loan_data,␣
 ↪percentage_missing_loan_data],axis = 1, keys=['Total', 'Percent'])
    return missingloan_data


missing_loan_data = missing_loan_data(loan_data)
missing_loan_data[missing_loan_data["Total"]>0]
```

[ ]:
```
                      Total   Percent
mort_acc              37795  9.543469
emp_title             22927  5.789208
emp_length            18301  4.621115
title                  1756  0.443401
pub_rec_bankruptcies    535  0.135091
revol_util              276  0.069692
```

4.4 Unique value counts wrt each column

[ ]:
```
#checking the unique values for columns
for i in loan_data.columns:
    print()
    print(f'Total Unique Values in {i} column are :- {loan_data[i].nunique()}')
    print(f'Unique Values in {i} column are :-\n {loan_data[i].unique()}')
    print(f'Value_counts of {i} column :-\n {loan_data[i].value_counts()}')
    print()
    print('-'*120)
```

```
Total Unique Values in loan_amnt column are :- 1397
```

```
Unique Values in loan_amnt column are :-
 [10000.  8000. 15600. … 36275. 36475.   725.]
Value_counts of loan_amnt column :-
 loan_amnt
10000.0    27668
12000.0    21366
15000.0    19903
20000.0    18969
35000.0    14576
           …
39200.0        1
38750.0        1
36275.0        1
36475.0        1
725.0          1
Name: count, Length: 1397, dtype: int64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in term column are :- 2
Unique Values in term column are :-
 [' 36 months' ' 60 months']
Value_counts of term column :-
 term
36 months    302005
60 months     94025
Name: count, dtype: int64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in int_rate column are :- 566
Unique Values in int_rate column are :-
 [11.44 11.99 10.49  6.49 17.27 13.33  5.32 11.14 10.99 16.29 13.11 14.64
  9.17 12.29  6.62  8.39 21.98  7.9   6.97  6.99 15.61 11.36 13.35 12.12
  9.99  8.19 18.75  6.03 14.99 16.78 13.67 13.98 16.99 19.91 17.86 21.49
 12.99 18.54  7.89 17.1  18.25 11.67  6.24  8.18 12.35 14.16 17.56 18.55
 22.15 10.39 15.99 16.07 24.99  9.67 19.19 21.   12.69 10.74  6.68 19.22
 11.49 16.55 19.97 24.7  13.49 18.24 16.49 25.78 25.83 18.64  7.51 13.99
 15.22 15.31  7.69 19.53 10.16  7.62  9.75 13.68 15.88 14.65  6.92 23.83
 10.75 18.49 20.31 17.57 27.31 19.99 22.99 12.59 10.37 14.33 13.53 22.45
 24.5  17.99  9.16 12.49 11.55 17.76 28.99 23.1  20.49 22.7  10.15  6.89
 19.52  8.9  14.3   9.49 25.99 24.08 13.05 14.98 16.59 11.26 25.89 14.48
 21.99 23.99  5.99 14.47 11.53  8.67  8.59 10.64 23.28 25.44  9.71 16.2
 19.24 24.11 15.8  15.96 14.49 18.99  5.79 19.29 14.54 14.09  9.25 19.05
 17.77 18.92 20.75 10.65 18.85 10.59 12.85 11.39 13.65 13.06  7.12 20.99
 13.61 12.73 14.46 16.24 25.49  7.39 10.78 20.8   7.88 15.95 12.39 21.18
```

```
 21.97 15.77  6.39 10.   12.53 13.43  7.49 25.57 21.48 18.39 11.47  7.26
 15.68 19.04 14.31 24.24  5.42 23.43 19.47  6.54 23.32 17.58 14.72  7.66
  9.76 13.23 13.48 12.42  9.8  11.71 14.27 21.15 22.95  8.49 17.74 15.59
 13.72  9.45  7.29 15.1  11.86 19.72 14.35 11.22 15.62 15.81 12.41 28.67
 11.48 13.66  9.91 23.76 17.14 18.84 12.23  6.17  8.94 14.22 19.03 25.29
  8.99  9.88 15.58 27.49  8.07 22.47 19.2  13.44 22.4  12.79 18.2  13.18
  7.24 14.84  5.93 15.28 13.85 25.28  8.    9.62 12.05 15.7  20.2  13.57
 21.67  7.4  25.8  12.68 11.83  7.37 11.11 14.85 16.   11.12 23.63  6.
  7.99  7.91 14.83 21.7  26.06 16.77 27.34 12.21  7.68 15.27 19.69  9.63
  7.14 20.5  16.02 12.84  7.74 15.33 19.79 22.2  18.62 17.49 16.89 15.21
 14.79 18.67  9.32 15.41 15.65 23.5  22.9  11.34 22.11 19.48 14.75 28.14
 13.22 23.4  23.13 28.18 12.88 22.06 24.49 16.45 21.6  28.49  8.38  6.76
 10.83 13.79  8.88 17.88 17.97 14.26  6.91 13.47  8.6  27.88  8.63 10.25
 14.91 12.74 10.96 25.88  7.43 16.4  20.25 24.89 12.87 20.16 14.17 12.18
 17.51 13.92 20.53 26.77 10.62 26.49 16.32 12.61 21.36 14.61 15.37 20.3
 14.59 16.7  19.89 10.95 18.17 18.21 17.93 22.39 24.83 13.8  19.42 23.7
  7.59 13.17 18.09 13.04 25.69  9.07 15.23 14.42 23.33 16.69 10.36 14.96
 10.38 26.24 24.2  12.98 20.85 13.36 26.57 23.52 22.78 13.16 15.13 25.11
 13.55 10.51 11.78  7.05 11.46 21.28 12.09 16.35  8.7  26.99 14.11 26.14
 16.82 23.26 18.79 10.28 19.36 18.3  17.06 17.19  7.75 17.34 20.89 22.35
 19.66 13.62 22.74 11.89 23.59  8.24 20.62 11.97 15.2  20.48 12.36 10.71
 25.09 20.11 27.79 29.49 11.58 19.13 11.66 13.75 30.74  9.38 27.99 11.59
  9.64 25.65  9.96 19.41 14.18 10.08 17.43 24.74 14.74 17.04 15.57 30.49
 17.8  10.91 14.82 29.96 12.92 12.22 15.45 11.72 10.2  14.7  20.69 15.05
 24.33 14.93 10.33 16.95 28.88 11.03 28.34 21.22 18.07  9.33 12.17 19.74
 20.9  20.03 17.39 29.67 12.04 23.22 10.01 22.48 24.76 13.3  20.77 10.14
 14.5  30.94  8.32 13.24 21.59 21.27 24.52 11.54 10.46 13.87 30.99  9.51
  9.83 19.39 12.86 30.79 21.74 11.09 16.11 17.26 22.85 18.91 18.43  9.2
 21.14 12.62 21.21 29.99 14.88 13.12 30.89 16.08 12.54 28.69 12.8  11.28
 23.91 22.94 19.16 20.86 11.63 19.82 11.41 21.82 12.72 20.4   9.7  18.72
 18.36 14.25 13.84 18.78 17.15 15.25 16.63 16.15 11.91 14.07  9.01 15.01
 21.64 15.83 18.53  7.42 12.67 15.76 16.33 30.84 13.93 14.12 14.28 20.17
 24.59 20.52 17.03 17.9  14.67 15.38 17.46 14.62 14.38 24.4  22.64 17.54
 17.44 15.07]
Value_counts of int_rate column :-
 int_rate
10.99    12411
12.99     9632
15.61     9350
11.99     8582
8.90      8019
         …
14.38        1
24.40        1
22.64        1
17.54        1
17.44        1
Name: count, Length: 566, dtype: int64
```

```
--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in installment column are :- 55706
Unique Values in installment column are :-
 [329.48 265.68 506.97 … 343.14 118.13 572.44]
Value_counts of installment column :-
 installment
327.34     968
332.10     791
491.01     736
336.90     686
392.81     683
          …
1146.14      1
218.49       1
961.66       1
569.10       1
555.96       1
Name: count, Length: 55706, dtype: int64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in grade column are :- 7
Unique Values in grade column are :-
 ['B' 'A' 'C' 'E' 'D' 'F' 'G']
Value_counts of grade column :-
 grade
B    116018
C    105987
A     64187
D     63524
E     31488
F     11772
G      3054
Name: count, dtype: int64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in sub_grade column are :- 35
Unique Values in sub_grade column are :-
 ['B4' 'B5' 'B3' 'A2' 'C5' 'C3' 'A1' 'B2' 'C1' 'A5' 'E4' 'A4' 'A3' 'D1'
 'C2' 'B1' 'D3' 'D5' 'D2' 'E1' 'E2' 'E5' 'F4' 'E3' 'D4' 'G1' 'F5' 'G2'
 'C4' 'F1' 'F3' 'G5' 'G4' 'F2' 'G3']
Value_counts of sub_grade column :-
```

```
 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

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in emp_title column are :- 173105
Unique Values in emp_title column are :-
 ['Marketing' 'Credit analyst ' 'Statistician' …
 "Michael's Arts & Crafts" 'licensed bankere' 'Gracon Services, Inc']
Value_counts of emp_title column :-
 emp_title
Teacher                         4389
```

```
Manager                          4250
Registered Nurse                 1856
RN                               1846
Supervisor                       1830
                                 …
Social Work/Care Manager            1
Regional Counsel                    1
Nor-Com Inc                         1
Director of the Bach Society        1
SPO II                              1
Name: count, Length: 173105, dtype: int64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in emp_length column are :- 11
Unique Values in emp_length column are :-
 ['10+ years' '4 years' '< 1 year' '6 years' '9 years' '2 years' '3 years'
 '8 years' '7 years' '5 years' '1 year' nan]
Value_counts of emp_length column :-
 emp_length
10+ years    126041
2 years       35827
< 1 year      31725
3 years       31665
5 years       26495
1 year        25882
4 years       23952
6 years       20841
7 years       20819
8 years       19168
9 years       15314
Name: count, dtype: int64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in home_ownership column are :- 6
Unique Values in home_ownership column are :-
 ['RENT' 'MORTGAGE' 'OWN' 'OTHER' 'NONE' 'ANY']
Value_counts of home_ownership column :-
 home_ownership
MORTGAGE    198348
RENT        159790
OWN          37746
OTHER          112
NONE            31
ANY              3
```

```
Name: count, dtype: int64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in annual_inc column are :- 27197
Unique Values in annual_inc column are :-
 [117000.    65000.    43057.   …  36111.    47212.    31789.88]
Value_counts of annual_inc column :-
 annual_inc
60000.0     15313
50000.0     13303
65000.0     11333
70000.0     10674
40000.0     10629
            …
42558.0         1
29899.0         1
25837.2         1
128647.0        1
23085.0         1
Name: count, Length: 27197, dtype: int64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in verification_status column are :- 3
Unique Values in verification_status column are :-
 ['Not Verified' 'Source Verified' 'Verified']
Value_counts of verification_status column :-
 verification_status
Verified           139563
Source Verified    131385
Not Verified       125082
Name: count, dtype: int64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in issue_d column are :- 115
Unique Values in issue_d column are :-
 ['Jan-2015' 'Nov-2014' 'Apr-2013' 'Sep-2015' 'Sep-2012' 'Oct-2014'
 'Apr-2012' 'Jun-2013' 'May-2014' 'Dec-2015' 'Apr-2015' 'Oct-2012'
 'Jul-2014' 'Feb-2013' 'Oct-2015' 'Jan-2014' 'Mar-2016' 'Apr-2014'
 'Jun-2011' 'Apr-2010' 'Jun-2014' 'Oct-2013' 'May-2013' 'Feb-2015'
 'Oct-2011' 'Jun-2015' 'Aug-2013' 'Feb-2014' 'Dec-2011' 'Mar-2013'
 'Jun-2016' 'Mar-2014' 'Nov-2013' 'Dec-2014' 'Apr-2016' 'Sep-2013'
 'May-2016' 'Jul-2015' 'Jul-2013' 'Aug-2014' 'May-2008' 'Mar-2010'
```

```
 'Dec-2013' 'Mar-2012' 'Mar-2015' 'Sep-2011' 'Jul-2012' 'Dec-2012'
 'Sep-2014' 'Nov-2012' 'Nov-2015' 'Jan-2011' 'May-2012' 'Feb-2016'
 'Jun-2012' 'Aug-2012' 'Jan-2016' 'May-2015' 'Oct-2016' 'Aug-2015'
 'Jul-2016' 'May-2009' 'Aug-2016' 'Jan-2012' 'Jan-2013' 'Nov-2010'
 'Jul-2011' 'Mar-2011' 'Feb-2012' 'May-2011' 'Aug-2010' 'Nov-2016'
 'Jul-2010' 'Sep-2010' 'Dec-2010' 'Feb-2011' 'Jun-2009' 'Aug-2011'
 'Dec-2016' 'Mar-2009' 'Jun-2010' 'May-2010' 'Nov-2011' 'Sep-2016'
 'Oct-2009' 'Mar-2008' 'Nov-2008' 'Dec-2009' 'Oct-2010' 'Sep-2009'
 'Oct-2007' 'Aug-2009' 'Jul-2009' 'Nov-2009' 'Jan-2010' 'Dec-2008'
 'Feb-2009' 'Oct-2008' 'Apr-2009' 'Feb-2010' 'Apr-2011' 'Apr-2008'
 'Aug-2008' 'Jan-2009' 'Feb-2008' 'Aug-2007' 'Sep-2008' 'Dec-2007'
 'Jan-2008' 'Sep-2007' 'Jun-2008' 'Jul-2008' 'Jun-2007' 'Nov-2007'
 'Jul-2007']
Value_counts of issue_d column :-
 issue_d
Oct-2014    14846
Jul-2014    12609
Jan-2015    11705
Dec-2013    10618
Nov-2013    10496
            …
Aug-2007       26
Sep-2008       25
Nov-2007       22
Sep-2007       15
Jun-2007        1
Name: count, Length: 115, dtype: int64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in loan_status column are :- 2
Unique Values in loan_status column are :-
 ['Fully Paid' 'Charged Off']
Value_counts of loan_status column :-
 loan_status
Fully Paid     318357
Charged Off     77673
Name: count, dtype: int64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in purpose column are :- 14
Unique Values in purpose column are :-
 ['vacation' 'debt_consolidation' 'credit_card' 'home_improvement'
 'small_business' 'major_purchase' 'other' 'medical' 'wedding' 'car'
 'moving' 'house' 'educational' 'renewable_energy']
```

```
Value_counts of purpose column :-
 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

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in title column are :- 48816
Unique Values in title column are :-
 ['Vacation' 'Debt consolidation' 'Credit card refinancing' …
 'Credit buster ' 'Loanforpayoff' 'Toxic Debt Payoff']
Value_counts of title column :-
 title
Debt consolidation                      152472
Credit card refinancing                  51487
Home improvement                         15264
Other                                    12930
Debt Consolidation                       11608
                                          …
Outboard Motor Repower Loan                  1
2011 Insurance and Debt Consolidation        1
Credit buster                                1
Loanforpayoff                                1
Toxic Debt Payoff                            1
Name: count, Length: 48816, dtype: int64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in dti column are :- 4262
Unique Values in dti column are :-
 [26.24 22.05 12.79 … 40.56 47.09 55.53]
Value_counts of dti column :-
 dti
```

```
0.00     313
14.40    310
19.20    302
16.80    301
18.00    300
        …
41.38      1
49.83      1
46.32      1
43.98      1
40.61      1
Name: count, Length: 4262, dtype: int64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in earliest_cr_line column are :- 684
Unique Values in earliest_cr_line column are :-
 ['Jun-1990' 'Jul-2004' 'Aug-2007' 'Sep-2006' 'Mar-1999' 'Jan-2005'
 'Aug-2005' 'Sep-1994' 'Jun-1994' 'Dec-1997' 'Dec-1990' 'May-1984'
 'Apr-1995' 'Jan-1997' 'May-2001' 'Mar-1982' 'Sep-1996' 'Jan-1990'
 'Mar-2000' 'Jan-2006' 'Oct-2006' 'Jan-2003' 'May-2008' 'Oct-2003'
 'Jun-2004' 'Jan-1999' 'Apr-1994' 'Apr-1998' 'Jul-2007' 'Apr-2002'
 'Oct-2007' 'Jun-2009' 'May-1997' 'Jul-2006' 'Sep-2003' 'Aug-1992'
 'Dec-1988' 'Feb-2002' 'Jan-1992' 'Aug-2001' 'Dec-2010' 'Oct-1999'
 'Sep-2004' 'Aug-1994' 'Jul-2003' 'Apr-2000' 'Dec-2004' 'Jun-1995'
 'Dec-2003' 'Jul-1994' 'Oct-1990' 'Dec-2001' 'Apr-1999' 'Feb-1995'
 'May-2003' 'Oct-2002' 'Mar-2004' 'Aug-2003' 'Oct-2000' 'Nov-2004'
 'Mar-2010' 'Mar-1996' 'May-1994' 'Jun-1996' 'Nov-1986' 'Jan-2001'
 'Jan-2002' 'Mar-2001' 'Sep-2012' 'Apr-2006' 'May-1998' 'Dec-2002'
 'Nov-2003' 'Oct-2005' 'May-1990' 'Jun-2003' 'Jun-2001' 'Jan-1998'
 'Oct-1978' 'Feb-2001' 'Jun-2006' 'Aug-1993' 'Apr-2001' 'Nov-2001'
 'Feb-2003' 'Jun-1993' 'Sep-1992' 'Nov-1992' 'Jun-1983' 'Oct-2001'
 'Jul-1999' 'Sep-1997' 'Nov-1993' 'Feb-1993' 'Apr-2007' 'Nov-1999'
 'Nov-2005' 'Dec-1992' 'Mar-1986' 'May-1989' 'Dec-2000' 'Mar-1991'
 'Mar-2005' 'Jun-2010' 'Dec-1998' 'Sep-2001' 'Nov-2000' 'Jan-1994'
 'Aug-2002' 'Jan-2011' 'Aug-2008' 'Jun-2005' 'Nov-1997' 'May-1996'
 'Apr-2010' 'May-1993' 'Sep-2005' 'Jun-1992' 'Apr-1986' 'Aug-1996'
 'Aug-1997' 'Jul-2005' 'May-2011' 'Sep-2002' 'Jan-1989' 'Aug-1999'
 'Feb-1992' 'Sep-1999' 'Jul-2001' 'May-1980' 'Oct-2008' 'Nov-2007'
 'Apr-1997' 'Jun-1986' 'Sep-1998' 'Jun-1982' 'Oct-1981' 'Feb-1994'
 'Dec-1984' 'Nov-1991' 'Nov-2006' 'Aug-2000' 'Oct-2004' 'Jun-2011'
 'Apr-1988' 'May-2004' 'Aug-1988' 'Mar-1994' 'Aug-2004' 'Dec-2006'
 'Nov-1998' 'Oct-1997' 'Mar-1989' 'Feb-1988' 'Jul-1982' 'Nov-1995'
 'Mar-1997' 'Oct-1994' 'Jul-1998' 'Jun-2002' 'May-1991' 'Oct-2011'
 'Sep-2007' 'Jan-2007' 'Jan-2010' 'Mar-1987' 'Feb-1997' 'Oct-1986'
 'Mar-2002' 'Jul-1993' 'Mar-2007' 'Aug-1989' 'Oct-1995' 'May-2007'
 'Dec-1993' 'Jun-1989' 'Apr-2004' 'Jun-1997' 'Apr-1996' 'Apr-1992'
```

```
'Oct-1998' 'Mar-1983' 'Mar-1985' 'Oct-1993' 'Feb-2000' 'Apr-2003'
'Oct-1985' 'Jul-1985' 'May-1978' 'Sep-2010' 'Oct-1996' 'Sep-2009'
'Jun-1999' 'Jan-2000' 'Sep-1987' 'Aug-1998' 'Jan-1995' 'Jul-1988'
'May-2000' 'Jun-1981' 'Feb-1998' 'Nov-1996' 'Aug-1967' 'Dec-1999'
'Aug-2006' 'Nov-2009' 'Jul-2000' 'Mar-1988' 'Jul-1992' 'Jul-1991'
'Mar-1990' 'May-1986' 'Jun-1991' 'Dec-1987' 'Jul-1996' 'Jul-1997'
'Aug-1990' 'Jan-1988' 'Dec-2005' 'Mar-2003' 'Feb-1999' 'Nov-1990'
'Jun-2000' 'Dec-1996' 'Jan-2004' 'May-1999' 'Sep-1972' 'Jul-1981'
'Sep-1993' 'Feb-2009' 'Nov-2002' 'Nov-1969' 'Jan-1993' 'May-2005'
'Sep-1982' 'Apr-1990' 'Feb-1996' 'Mar-1993' 'Apr-1978' 'Jul-1995'
'May-1995' 'Apr-1991' 'Mar-1998' 'Aug-1991' 'Jul-2002' 'Oct-1989'
'Apr-1984' 'Dec-2009' 'Sep-2000' 'Jan-1982' 'Jun-1998' 'Jan-1996'
'Nov-1987' 'May-2010' 'Jul-1989' 'Jun-1987' 'Oct-1987' 'Aug-1995'
'Feb-2004' 'Oct-1991' 'Dec-1989' 'Oct-1992' 'Feb-2005' 'Apr-1993'
'Dec-1985' 'Sep-1979' 'Feb-2007' 'Nov-1989' 'Apr-2005' 'Mar-1978'
'Sep-1985' 'Nov-1994' 'Jun-2008' 'Apr-1987' 'Dec-1983' 'Dec-2007'
'May-1979' 'May-1992' 'Jul-1990' 'Mar-1995' 'Feb-2006' 'Feb-1985'
'Sep-1989' 'Aug-2009' 'Nov-2008' 'Nov-1981' 'Jan-2008' 'Aug-1987'
'Nov-1985' 'Dec-1965' 'Sep-1995' 'Jan-1986' 'Oct-2009' 'May-2002'
'Aug-1980' 'Sep-1977' 'Sep-1988' 'Oct-1984' 'May-1988' 'Aug-1984'
'Nov-1988' 'May-1974' 'Nov-1982' 'Oct-1983' 'Sep-1991' 'Feb-1984'
'Feb-1991' 'Jan-1981' 'Jun-1985' 'Dec-1976' 'Dec-1994' 'Dec-1980'
'Sep-1984' 'Jun-2007' 'Aug-1979' 'Sep-2008' 'Apr-1983' 'Mar-2006'
'Jun-1984' 'Jul-1984' 'Jan-1985' 'Dec-1995' 'Apr-2008' 'Mar-2008'
'Jan-1983' 'Dec-1986' 'Jun-1979' 'Dec-1975' 'Nov-1983' 'Jul-1986'
'Nov-1977' 'Dec-1982' 'May-1985' 'Feb-1983' 'Aug-1982' 'Oct-1980'
'Mar-1979' 'Jan-1978' 'Mar-1984' 'May-1983' 'Jul-2008' 'Apr-1982'
'Jul-1983' 'Feb-1990' 'Dec-2008' 'Jul-1975' 'Dec-1971' 'Feb-2008'
'Mar-2011' 'Feb-1987' 'Feb-1989' 'Aug-1985' 'Jul-2010' 'Apr-1989'
'Feb-1980' 'May-2006' 'Nov-2010' 'Apr-2009' 'Feb-2010' 'May-1976'
'Feb-1981' 'Jan-2012' 'Oct-1988' 'Nov-1984' 'May-1982' 'Oct-1975'
'Jun-1988' 'May-1972' 'Apr-2013' 'Sep-1990' 'Oct-1982' 'Feb-2013'
'Mar-1992' 'Aug-1981' 'Feb-2011' 'Nov-1974' 'Feb-1978' 'Sep-1983'
'Jul-2011' 'Nov-1979' 'Aug-1983' 'Apr-1985' 'Jul-2009' 'Jan-1971'
'Jul-1987' 'Aug-1978' 'Aug-2010' 'Oct-1976' 'Aug-1986' 'Jan-1991'
'Dec-1991' 'May-2009' 'Aug-2011' 'Jun-1964' 'Jan-1974' 'May-1981'
'Jun-1972' 'Jun-1978' 'Sep-1986' 'Jan-1987' 'Jan-1975' 'Feb-1982'
'Jan-1980' 'Feb-1977' 'Sep-1980' 'Nov-1978' 'Jul-1974' 'Jun-1970'
'Jan-1984' 'Nov-1980' 'May-1987' 'Sep-1970' 'Jan-1976' 'Feb-1986'
'Oct-2010' 'Apr-1979' 'Oct-1979' 'Jan-1979' 'Sep-2011' 'Jul-1979'
'Sep-1975' 'Mar-1981' 'Aug-1971' 'Apr-1980' 'Apr-1977' 'Jan-1965'
'Nov-1976' 'Nov-1970' 'Nov-2011' 'Nov-1973' 'Sep-1981' 'Jul-1980'
'Mar-2012' 'Dec-1974' 'Mar-1977' 'Dec-1977' 'May-2012' 'Dec-1979'
'Jan-2009' 'Jan-1970' 'Dec-2011' 'Feb-1979' 'Mar-1976' 'Jan-1973'
'Oct-1973' 'Mar-1969' 'Oct-1977' 'Mar-1975' 'Aug-1977' 'Jun-1969'
'Oct-1963' 'Nov-1960' 'Aug-1970' 'Feb-1975' 'Sep-1974' 'May-1966'
'Apr-1972' 'Apr-1973' 'Apr-2012' 'May-1975' 'Sep-1966' 'Feb-1969'
'Feb-2012' 'Jan-1961' 'Aug-1973' 'Feb-1972' 'Apr-1975' 'Jul-1978'
```

```
 'Oct-1970' 'Mar-1980' 'Sep-1976' 'Apr-2011' 'Nov-2012' 'Aug-1976'
 'Jun-1975' 'Apr-1981' 'Mar-2009' 'Jun-1977' 'Apr-1971' 'Sep-1969'
 'Jun-2012' 'Apr-1976' 'Feb-1965' 'Jul-1977' 'Jun-1976' 'Mar-1973'
 'Oct-1972' 'Dec-1978' 'Nov-1967' 'Sep-1967' 'Nov-1971' 'Jun-1980'
 'May-1964' 'Feb-1971' 'May-1970' 'Apr-1970' 'Mar-1971' 'Apr-1969'
 'Jan-1963' 'Jun-1974' 'Oct-1974' 'May-1977' 'Dec-1981' 'Jan-1969'
 'Feb-1976' 'Mar-1970' 'Aug-1968' 'Feb-1970' 'Jun-1971' 'Jun-1963'
 'Jun-2013' 'Mar-1972' 'Aug-2012' 'Jan-1967' 'Feb-1968' 'Dec-1969'
 'Jan-1977' 'Jul-1970' 'Feb-1973' 'Mar-1974' 'Feb-1974' 'Dec-1960'
 'Jul-1972' 'Jul-1973' 'Sep-1964' 'Jul-1965' 'Oct-1958' 'Jul-2012'
 'Jun-1973' 'Sep-1978' 'Nov-1975' 'Jul-1963' 'Jan-1964' 'Dec-1968'
 'May-1958' 'Sep-1973' 'May-1971' 'Dec-1972' 'Aug-1965' 'Jul-1976'
 'Oct-2012' 'May-1973' 'Apr-1955' 'Apr-1966' 'Jan-1968' 'Nov-1968'
 'Oct-1969' 'Mar-2013' 'Jan-2013' 'Jul-1967' 'Oct-1965' 'Jan-1966'
 'Aug-1972' 'Jul-1969' 'May-1965' 'Jan-1953' 'Aug-1974' 'May-1968'
 'Aug-1969' 'May-2013' 'Oct-1967' 'Aug-1975' 'Apr-1974' 'Sep-1971'
 'Apr-1968' 'Jul-1971' 'Jan-1972' 'Nov-1965' 'Dec-1970' 'Dec-1973'
 'Nov-1972' 'Oct-1959' 'Oct-1962' 'Apr-1967' 'Oct-1971' 'Nov-1963'
 'Oct-1968' 'Dec-1962' 'Jun-1960' 'Jan-1960' 'Sep-2013' 'May-1969'
 'Dec-1966' 'Feb-1967' 'Dec-1967' 'Aug-1961' 'Sep-1968' 'Oct-1964'
 'Aug-1966' 'Jul-1966' 'Apr-1964' 'Sep-1962' 'Jul-2013' 'Jun-1967'
 'Apr-1965' 'Jun-1966' 'Jan-1955' 'Jan-1962' 'Feb-1964' 'Aug-1958'
 'Jul-1968' 'May-1967' 'Dec-1959' 'Sep-1963' 'Dec-2012' 'Dec-1963'
 'Jan-1944' 'Jun-1965' 'May-1962' 'Mar-1967' 'Mar-1968' 'Jan-1956'
 'Sep-1965' 'Dec-1951' 'Aug-2013' 'Jun-1968' 'Mar-1965' 'Oct-1957'
 'Nov-1966' 'Dec-1958' 'Feb-1957' 'Feb-1963' 'Mar-1963' 'Jan-1959'
 'May-1955' 'Feb-1966' 'Nov-1950' 'Mar-1964' 'Jan-1958' 'Nov-1964'
 'Sep-1961' 'Apr-1963' 'Jul-1964' 'Nov-1955' 'Jun-1957' 'Dec-1964'
 'Nov-1953' 'Apr-1961' 'Mar-1966' 'Oct-1960' 'Jul-1959' 'Jul-1961'
 'Jan-1954' 'Dec-1956' 'Mar-1962' 'Jul-1960' 'Sep-1959' 'Dec-1950'
 'Oct-1966' 'Apr-1960' 'Jul-1958' 'Nov-1954' 'Nov-1957' 'Jun-1962'
 'May-1963' 'Jul-1955' 'Oct-1950' 'Dec-1961' 'Aug-1951' 'Oct-2013'
 'Aug-1964' 'Apr-1962' 'Jun-1955' 'Jul-1962' 'Jan-1957' 'Nov-1958'
 'Jul-1951' 'Nov-1959' 'Apr-1958' 'Mar-1960' 'Sep-1957' 'Nov-1961'
 'Sep-1960' 'May-1959' 'Jun-1959' 'Feb-1962' 'Sep-1956' 'Aug-1960'
 'Feb-1961' 'Jan-1948' 'Aug-1963' 'Oct-1961' 'Aug-1962' 'Aug-1959']
Value_counts of earliest_cr_line column :-
 earliest_cr_line
Oct-2000    3017
Aug-2000    2935
Oct-2001    2896
Aug-2001    2884
Nov-2000    2736
            …
Feb-1957       1
Nov-1950       1
May-1955       1
Sep-1961       1
```

```
Nov-1955       1
Name: count, Length: 684, dtype: int64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in open_acc column are :- 61
Unique Values in open_acc column are :-
 [16. 17. 13.  6.  8. 11.  5. 30.  9. 15. 12. 10. 18.  7.  4. 14. 20. 19.
 21. 23.  3. 26. 42. 22. 25. 28.  2. 34. 24. 27. 31. 32. 33.  1. 29. 36.
 40. 35. 37. 41. 44. 39. 49. 48. 38. 51. 50. 43. 46.  0. 47. 57. 53. 58.
 52. 54. 45. 90. 56. 55. 76.]
Value_counts of open_acc column :-
 open_acc
9.0     36779
10.0    35441
8.0     35137
11.0    32695
7.0     31328
        …
56.0        2
55.0        2
57.0        1
58.0        1
90.0        1
Name: count, Length: 61, dtype: int64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in pub_rec column are :- 20
Unique Values in pub_rec column are :-
 [ 0.  1.  2.  3.  4.  6.  5.  8.  9. 10. 11.  7. 19. 13. 40. 17. 86. 12.
 24. 15.]
Value_counts of pub_rec column :-
 pub_rec
0.0     338272
1.0      49739
2.0       5476
3.0       1521
4.0        527
5.0        237
6.0        122
7.0         56
8.0         34
9.0         12
10.0        11
11.0         8
```

```
13.0         4
12.0         4
19.0         2
40.0         1
17.0         1
86.0         1
24.0         1
15.0         1
Name: count, dtype: int64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in revol_bal column are :- 55622
Unique Values in revol_bal column are :-
 [ 36369.  20131.  11987. …  34531. 151912.  29244.]
Value_counts of revol_bal column :-
 revol_bal
0.0         2128
5655.0        41
7792.0        38
6095.0        38
3953.0        37
             …
147559.0       1
50316.0        1
222641.0       1
568659.0       1
57725.0        1
Name: count, Length: 55622, dtype: int64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in revol_util column are :- 1226
Unique Values in revol_util column are :-
 [ 41.8   53.3   92.2  …  56.26 111.4  128.1 ]
Value_counts of revol_util column :-
 revol_util
0.00      2213
53.00      752
60.00      739
61.00      734
55.00      730
           …
146.10       1
109.30       1
108.10       1
```

```
115.30       1
37.63        1
Name: count, Length: 1226, dtype: int64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in total_acc column are :- 118
Unique Values in total_acc column are :-
 [ 25.  27.  26.  13.  43.  23.  15.  40.  37.  61.  35.  22.  20.  36.
  38.   7.  18.  10.  17.  29.  16.  21.  34.   9.  14.  59.  41.  19.
  12.  30.  56.  24.  28.   8.  52.  31.  44.  39.  50.  11.  62.  32.
   5.  33.  46.  42.   6.  49.  45.  57.  48.  67.  47.  51.  58.   3.
  55.  63.  53.   4.  71.  69.  54.  64.  81.  72.  60.  68.  65.  73.
  78.  84.   2.  76.  75.  79.  87.  77. 104.  89.  70. 105.  97.  66.
 108.  74.  80.  82.  91.  93. 106.  90.  85.  88.  83. 111.  86. 101.
 135.  92.  94.  95.  99. 102. 129. 110. 124. 151. 107. 118. 150. 115.
 117.  96.  98. 100. 116. 103.]
Value_counts of total_acc column :-
 total_acc
21.0     14280
22.0     14260
20.0     14228
23.0     13923
24.0     13878
         …
150.0        1
117.0        1
115.0        1
100.0        1
103.0        1
Name: count, Length: 118, dtype: int64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in initial_list_status column are :- 2
Unique Values in initial_list_status column are :-
 ['w' 'f']
Value_counts of initial_list_status column :-
 initial_list_status
f    238066
w    157964
Name: count, dtype: int64

--------------------------------------------------------------------------------
----------------------------------------
```

```
Total Unique Values in application_type column are :- 3
Unique Values in application_type column are :-
 ['INDIVIDUAL' 'JOINT' 'DIRECT_PAY']
Value_counts of application_type column :-
 application_type
INDIVIDUAL    395319
JOINT            425
DIRECT_PAY       286
Name: count, dtype: int64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in mort_acc column are :- 33
Unique Values in mort_acc column are :-
 [ 0.  3.  1.  4.  2.  6.  5. nan 10.  7. 12. 11.  8.  9. 13. 14. 22. 34.
 15. 25. 19. 16. 17. 32. 18. 24. 21. 20. 31. 28. 30. 23. 26. 27.]
Value_counts of mort_acc column :-
 mort_acc
0.0     139777
1.0      60416
2.0      49948
3.0      38049
4.0      27887
5.0      18194
6.0      11069
7.0       6052
8.0       3121
9.0       1656
10.0       865
11.0       479
12.0       264
13.0       146
14.0       107
15.0        61
16.0        37
17.0        22
18.0        18
19.0        15
20.0        13
24.0        10
22.0         7
21.0         4
25.0         4
27.0         3
26.0         2
32.0         2
31.0         2
```

```
23.0         2
34.0         1
28.0         1
30.0         1
Name: count, dtype: int64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in pub_rec_bankruptcies column are :- 9
Unique Values in pub_rec_bankruptcies column are :-
 [ 0.  1.  2.  3. nan  4.  5.  6.  7.  8.]
Value_counts of pub_rec_bankruptcies column :-
 pub_rec_bankruptcies
0.0    350380
1.0     42790
2.0      1847
3.0       351
4.0        82
5.0        32
6.0         7
7.0         4
8.0         2
Name: count, dtype: int64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in address column are :- 393700
Unique Values in address column are :-
 ['0174 Michelle Gateway\r\nMendozaberg, OK 22690'
 '1076 Carney Fort Apt. 347\r\nLoganmouth, SD 05113'
 '87025 Mark Dale Apt. 269\r\nNew Sabrina, WV 05113' …
 '953 Matthew Points Suite 414\r\nReedfort, NY 70466'
 '7843 Blake Freeway Apt. 229\r\nNew Michael, FL 29597'
 '787 Michelle Causeway\r\nBriannaton, AR 48052']
Value_counts of address column :-
 address
USS Smith\r\nFPO AP 70466                                  8
USNS Johnson\r\nFPO AE 05113                               8
USCGC Smith\r\nFPO AE 70466                                8
USS Johnson\r\nFPO AE 48052                                8
USNS Johnson\r\nFPO AP 48052                               7
                                                          ..
12951 Williams Crossing\r\nJohnnyville, DC 30723           1
0114 Fowler Field Suite 028\r\nRachelborough, LA 05113     1
953 Matthew Points Suite 414\r\nReedfort, NY 70466         1
7843 Blake Freeway Apt. 229\r\nNew Michael, FL 29597       1
```

```
823 Reid Ford\r\nDelacruzside, MA 00813                    1
Name: count, Length: 393700, dtype: int64
```

Observations: - The dataset has 396030 rows and 27 columns. With 12 columns with numerical values and 15 with object datatype. - `Target variable`: **Loan Status** - issue_d, earliest_cr_line are datetime features„ form which month and year could be extracted. - 15% rows are having null values. 15% is higher value. Should not delete them. Should handle the null value properly. - May be it is possible to impute some of the null values of title using purpose column as both columns are similar in nature. - emp title null can be modified to UNKNOWN title - As Loan Tap main motto is to provide as many loans as possible (even it is less secure), pub_rec_bankruptcies can be treated as 0.

5 Missing Value Imputation

[ ]:
```
loan_data['emp_title'].fillna('Unknown', inplace = True)
```

```
C:\Users\sreem\AppData\Local\Temp\ipykernel_34316\2501973861.py:1:
FutureWarning: A value is trying to be set on a copy of a DataFrame or Series
through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  loan_data['emp_title'].fillna('Unknown', inplace = True)
```

[ ]:
```
Null_replace_dict_purpose_title = {}
for i in loan_data["purpose"].unique():
    Null_replace_dict_purpose_title[i] = loan_data[loan_data["purpose"] ==␣
 ↪i]["title"].mode()[0]
print(Null_replace_dict_purpose_title)
```

```
{'vacation': 'Vacation', 'debt_consolidation': 'Debt consolidation',
'credit_card': 'Credit card refinancing', 'home_improvement': 'Home
improvement', 'small_business': 'Business', 'major_purchase': 'Major purchase',
'other': 'Other', 'medical': 'Medical expenses', 'wedding': 'Wedding Loan',
'car': 'Car financing', 'moving': 'Moving and relocation', 'house': 'Home
buying', 'educational': 'Student Loan', 'renewable_energy': 'Green loan'}
```

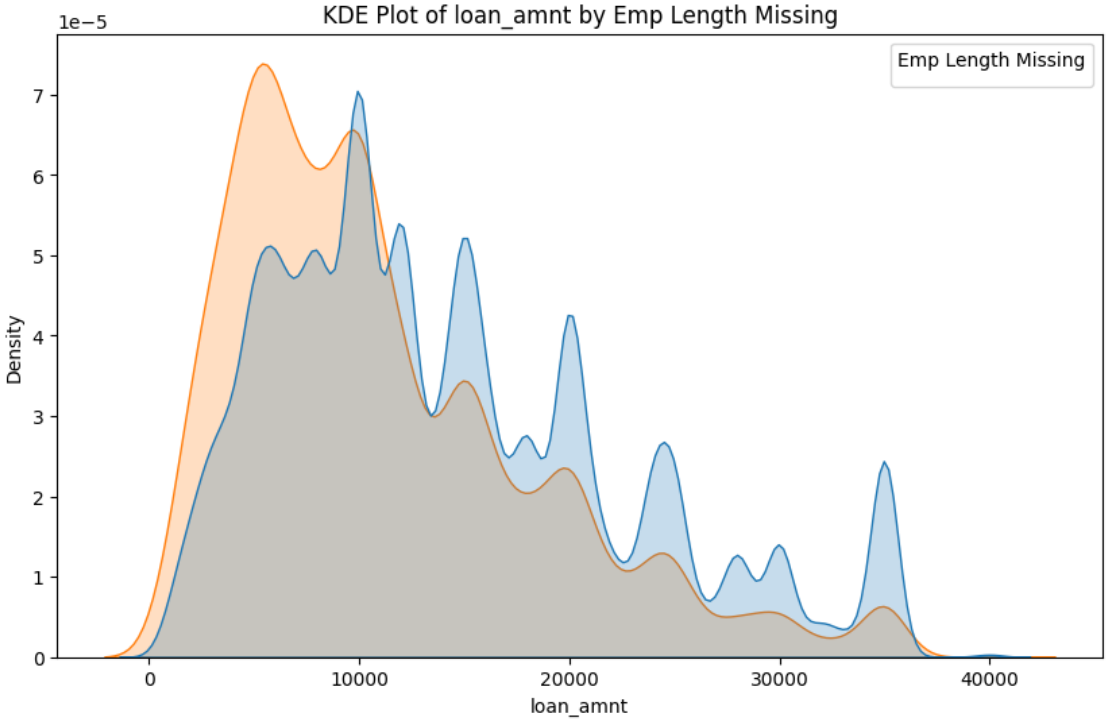
[ ]:
```
loan_data["title"] = loan_data["title"].fillna(loan_data["purpose"].
 ↪map(Null_replace_dict_purpose_title))
```

5.0.1 Checking if emp_lenght is related to any other feature using kde and count plots.

[ ]:
```
# Step 1: Create a new column indicating if emp_length is missing
loan_data['emp_length_missing'] = loan_data['emp_length'].isna().map({True:␣
 ↪'Missing', False: 'Present'})

# Step 2: Separate numerical and categorical columns
numerical_cols = loan_data.select_dtypes(include=['float64', 'int64']).columns.
 ↪tolist()
categorical_cols = loan_data.select_dtypes(include=['object']).columns.tolist()

# We don't want to include the newly added or unnecessary columns
categorical_cols = [col for col in categorical_cols if col not in ['address',␣
 ↪'emp_title', 'title', 'emp_length', 'emp_length_missing']]

# Step 3: KDE plots for numerical columns
for col in numerical_cols:
    plt.figure(figsize=(10,6))
    ax = sns.kdeplot(
        data=loan_data,
        x=col,
        hue='emp_length_missing',
        fill=True,
        common_norm=False,  # Normalize separately for Missing and Present
        bw_adjust=1         # Controls smoothness (optional)
    )

    # No need to multiply y_labels by 100 manually now
    # Set y-axis label to Percentage (conceptual - because KDE shows␣
 ↪probability)
    plt.title(f'KDE Plot of {col} by Emp Length Missing')
    plt.xlabel(col)
    plt.ylabel('Density')  # Density area is 1 for each group separately
    plt.legend(title='Emp Length Missing')
    plt.show()

# Step 4: Correct Count plots with Percentage
for col in categorical_cols:
    plt.figure(figsize=(12,6))

    # Calculate percentage manually for correct scaling
    temp_df = loan_data.groupby([col, 'emp_length_missing']).size().
 ↪reset_index(name='count')
```

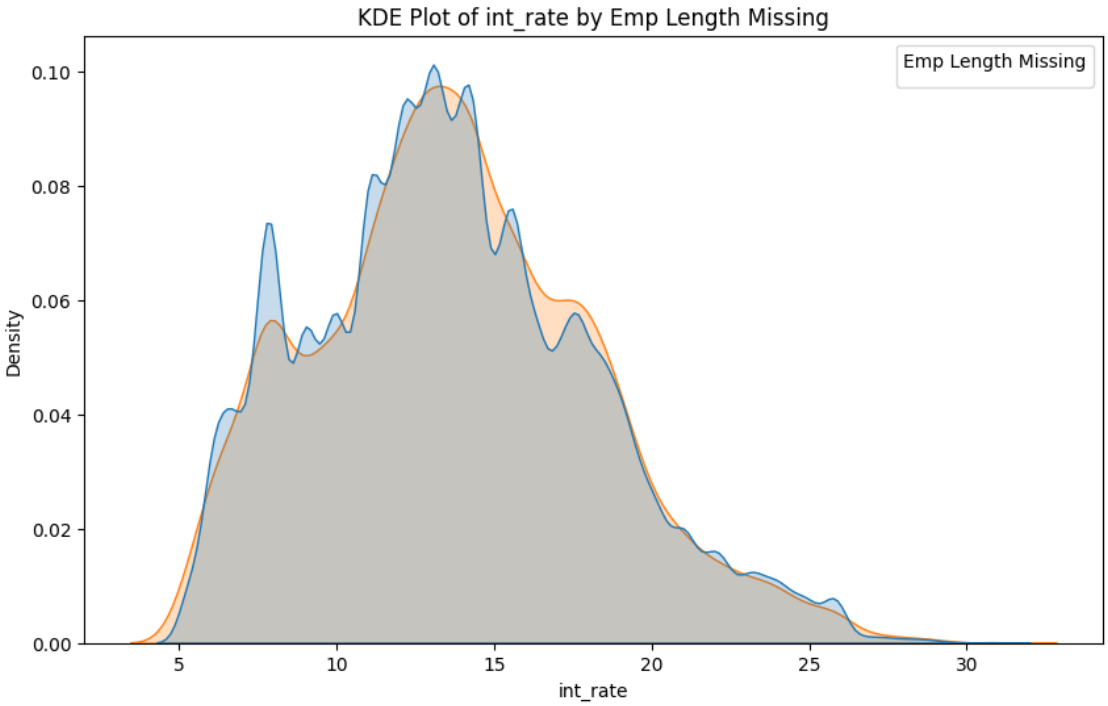
```
    total_counts = temp_df.groupby('emp_length_missing')['count'].
↪transform('sum')
    temp_df['percentage'] = (temp_df['count'] / total_counts) * 100

    ax = sns.barplot(
        data=temp_df,
        x=col,
        y='percentage',
        hue='emp_length_missing'
    )

    plt.title(f'Percentage Count Plot of {col} by Emp Length Missing')
    plt.xlabel(col)
    plt.ylabel('Percentage')
    plt.xticks(rotation=45, ha='right')
    plt.legend(title='Emp Length Missing')
    plt.show()
```

```
C:\Users\sreem\AppData\Local\Temp\ipykernel_34316\3191714987.py:28: UserWarning:
No artists with labels found to put in legend.  Note that artists whose label
start with an underscore are ignored when legend() is called with no argument.
  plt.legend(title='Emp Length Missing')
```

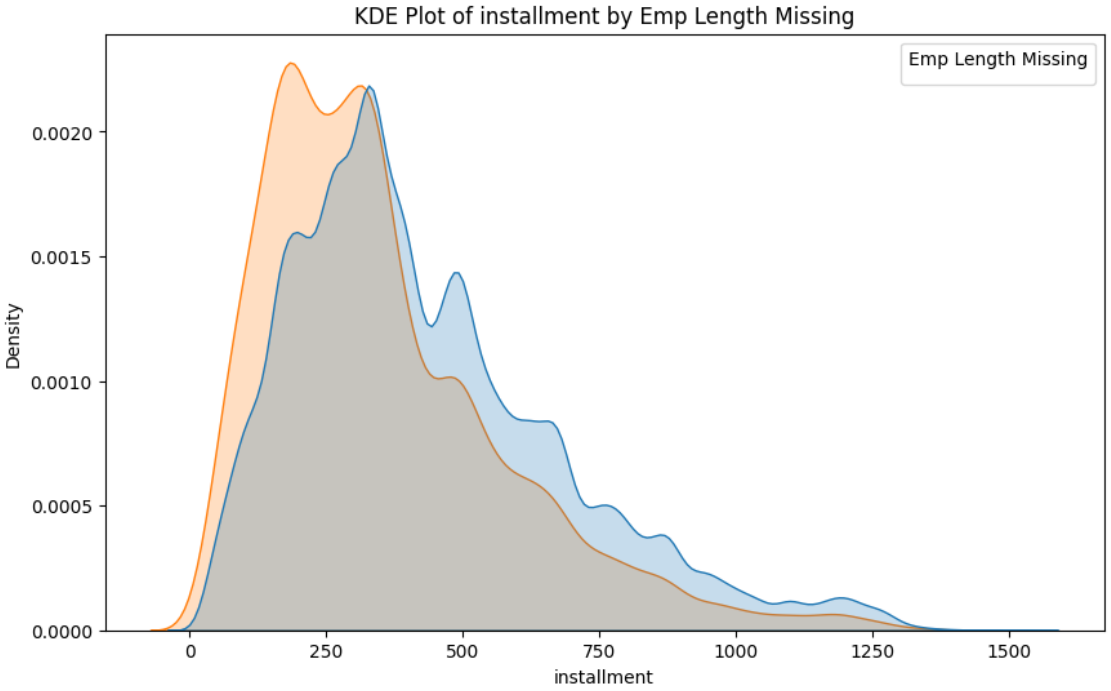



```
C:\Users\sreem\AppData\Local\Temp\ipykernel_34316\3191714987.py:28: UserWarning:
```

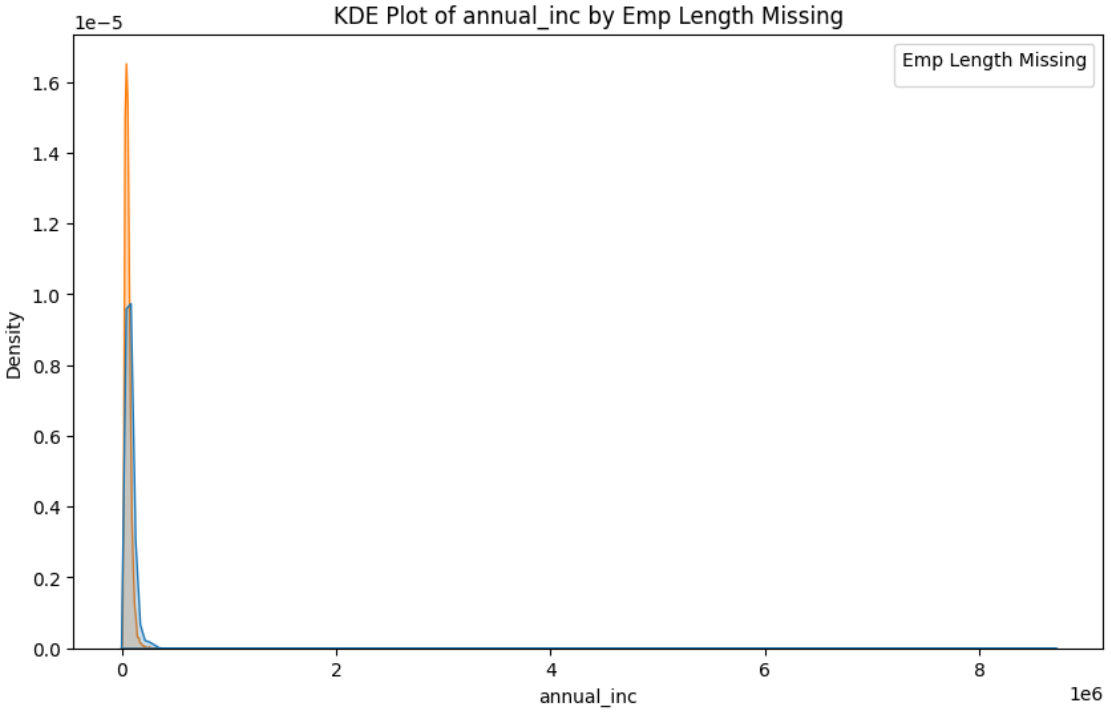
```
No artists with labels found to put in legend.  Note that artists whose label
start with an underscore are ignored when legend() is called with no argument.
  plt.legend(title='Emp Length Missing')
```



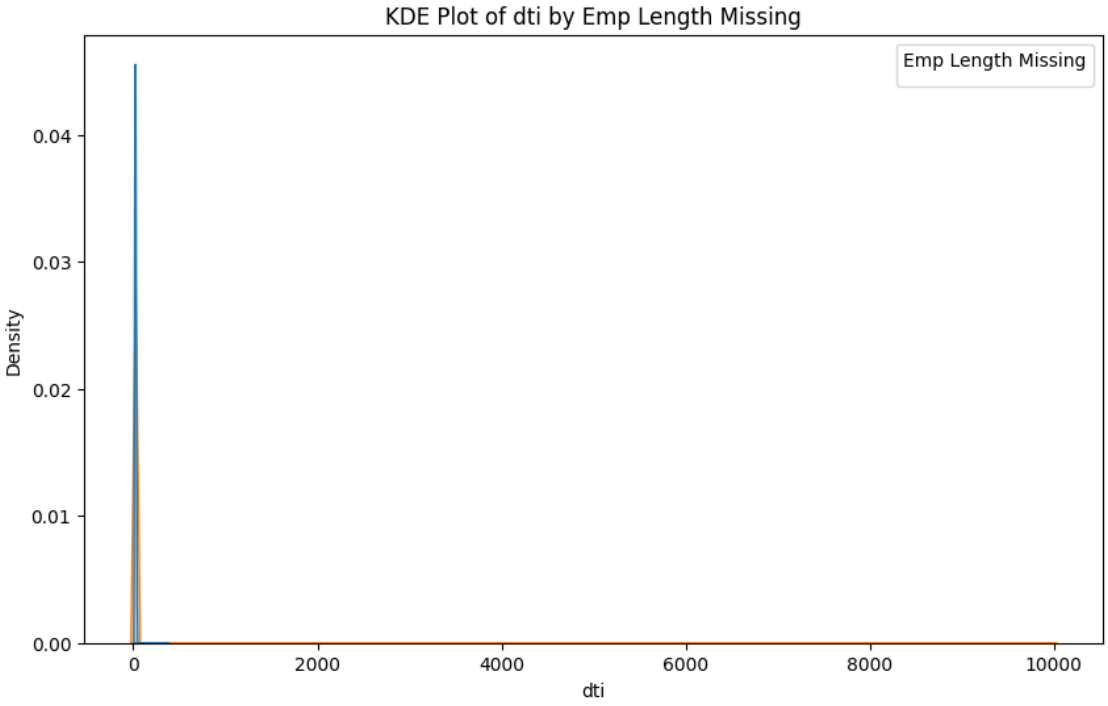

```
C:\Users\sreem\AppData\Local\Temp\ipykernel_34316\3191714987.py:28: UserWarning:
No artists with labels found to put in legend.  Note that artists whose label
start with an underscore are ignored when legend() is called with no argument.
  plt.legend(title='Emp Length Missing')
```



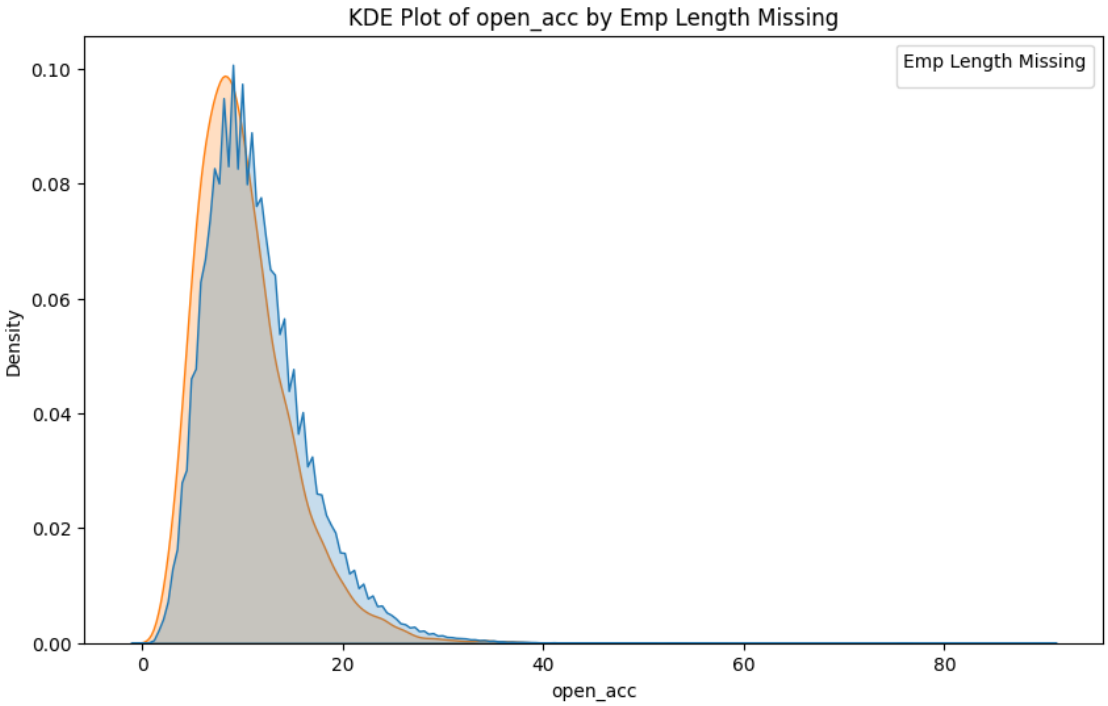

```
C:\Users\sreem\AppData\Local\Temp\ipykernel_34316\3191714987.py:28: UserWarning:
No artists with labels found to put in legend.  Note that artists whose label
start with an underscore are ignored when legend() is called with no argument.
  plt.legend(title='Emp Length Missing')
```



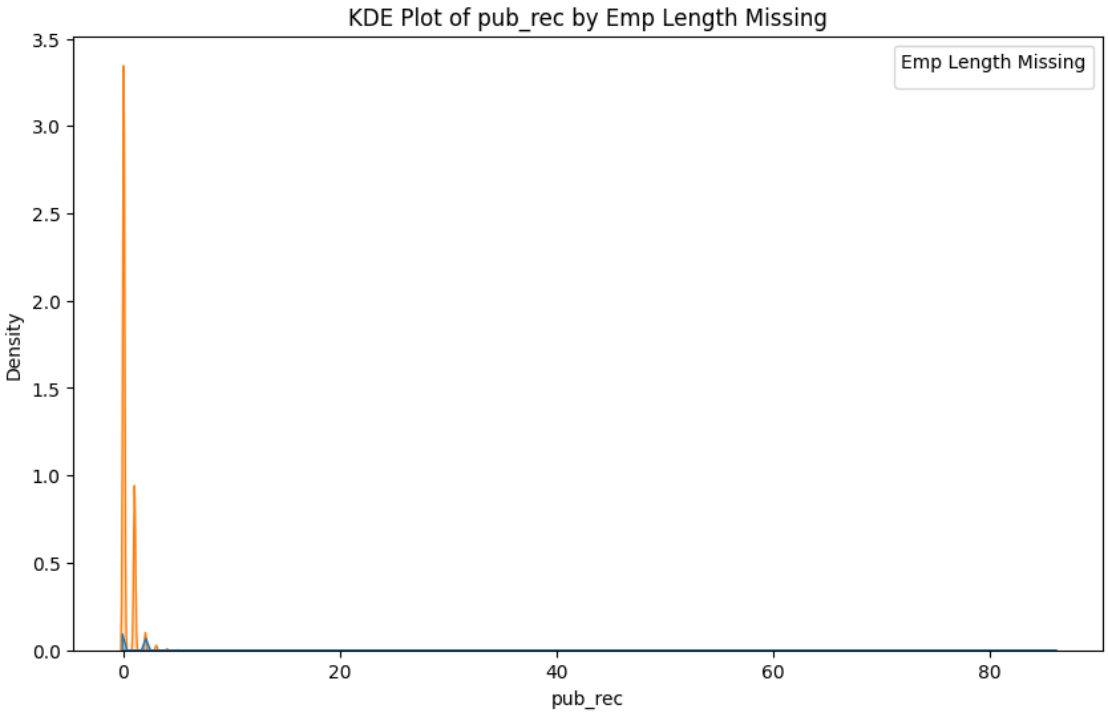
```
C:\Users\sreem\AppData\Local\Temp\ipykernel_34316\3191714987.py:28: UserWarning:
No artists with labels found to put in legend.  Note that artists whose label
start with an underscore are ignored when legend() is called with no argument.
  plt.legend(title='Emp Length Missing')
```



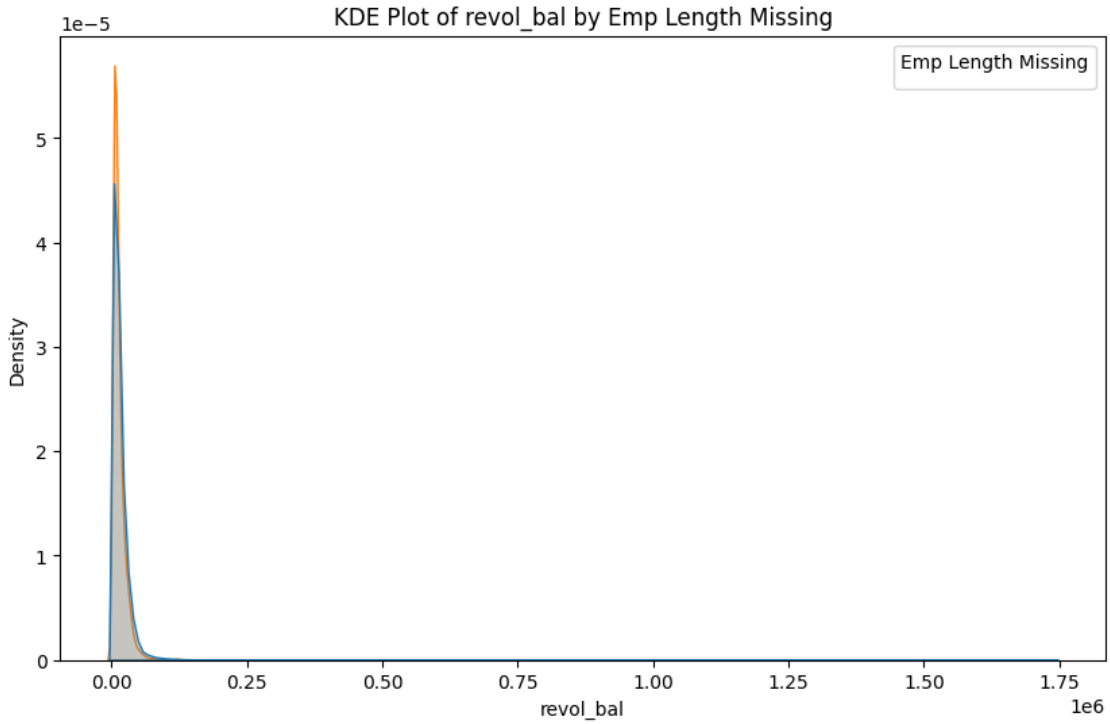

```
C:\Users\sreem\AppData\Local\Temp\ipykernel_34316\3191714987.py:28: UserWarning:
No artists with labels found to put in legend.  Note that artists whose label
start with an underscore are ignored when legend() is called with no argument.
  plt.legend(title='Emp Length Missing')
```



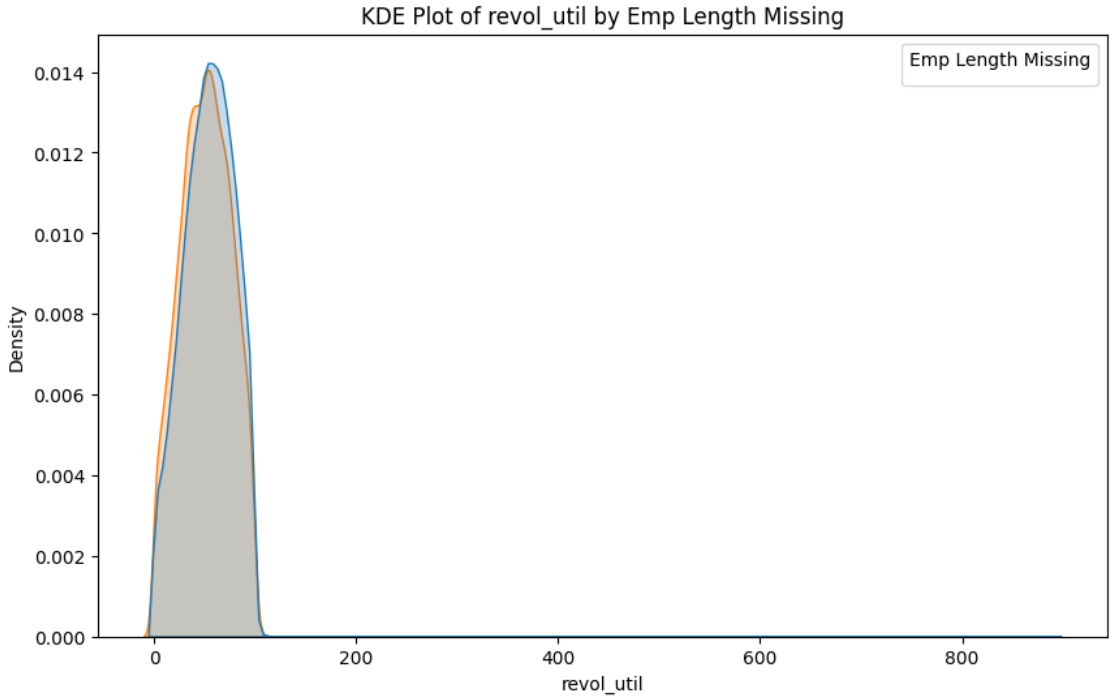

```
C:\Users\sreem\AppData\Local\Temp\ipykernel_34316\3191714987.py:28: UserWarning:
No artists with labels found to put in legend.  Note that artists whose label
start with an underscore are ignored when legend() is called with no argument.
  plt.legend(title='Emp Length Missing')
```



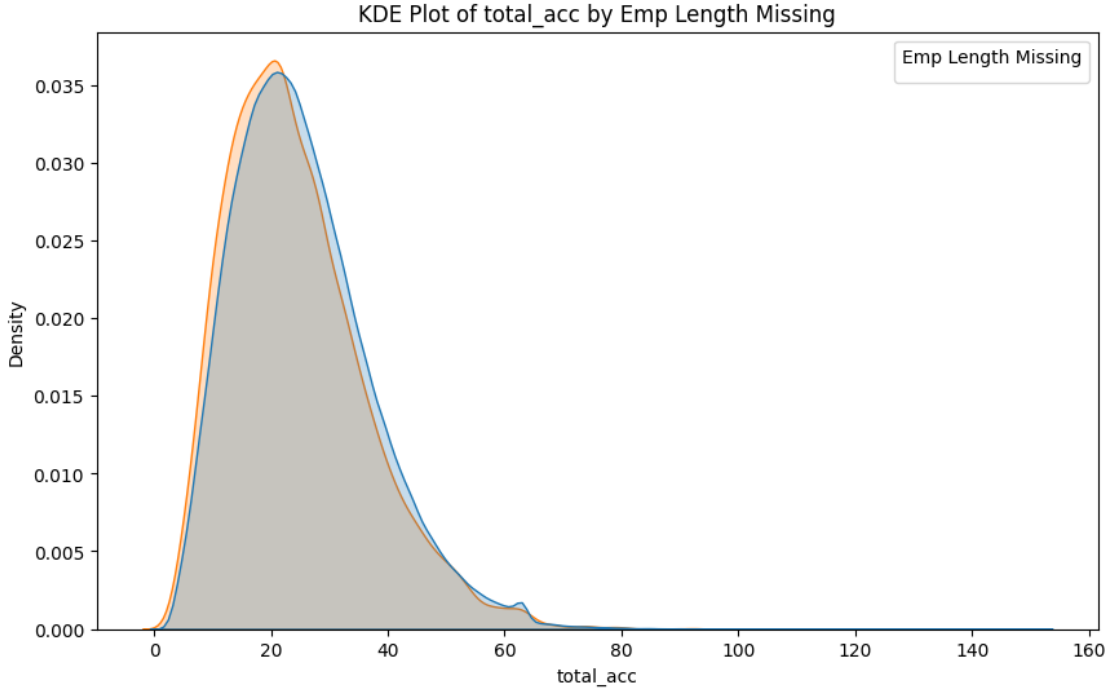

```
C:\Users\sreem\AppData\Local\Temp\ipykernel_34316\3191714987.py:28: UserWarning:
No artists with labels found to put in legend.  Note that artists whose label
start with an underscore are ignored when legend() is called with no argument.
  plt.legend(title='Emp Length Missing')
```



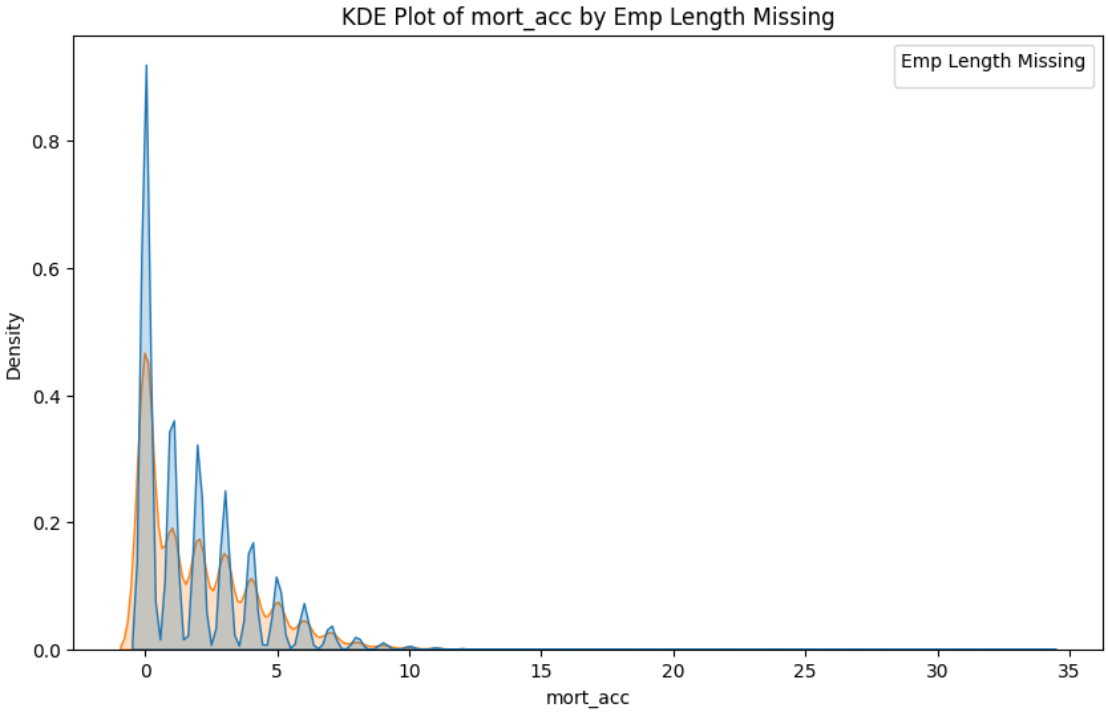

```
C:\Users\sreem\AppData\Local\Temp\ipykernel_34316\3191714987.py:28: UserWarning:
No artists with labels found to put in legend.  Note that artists whose label
start with an underscore are ignored when legend() is called with no argument.
  plt.legend(title='Emp Length Missing')
```



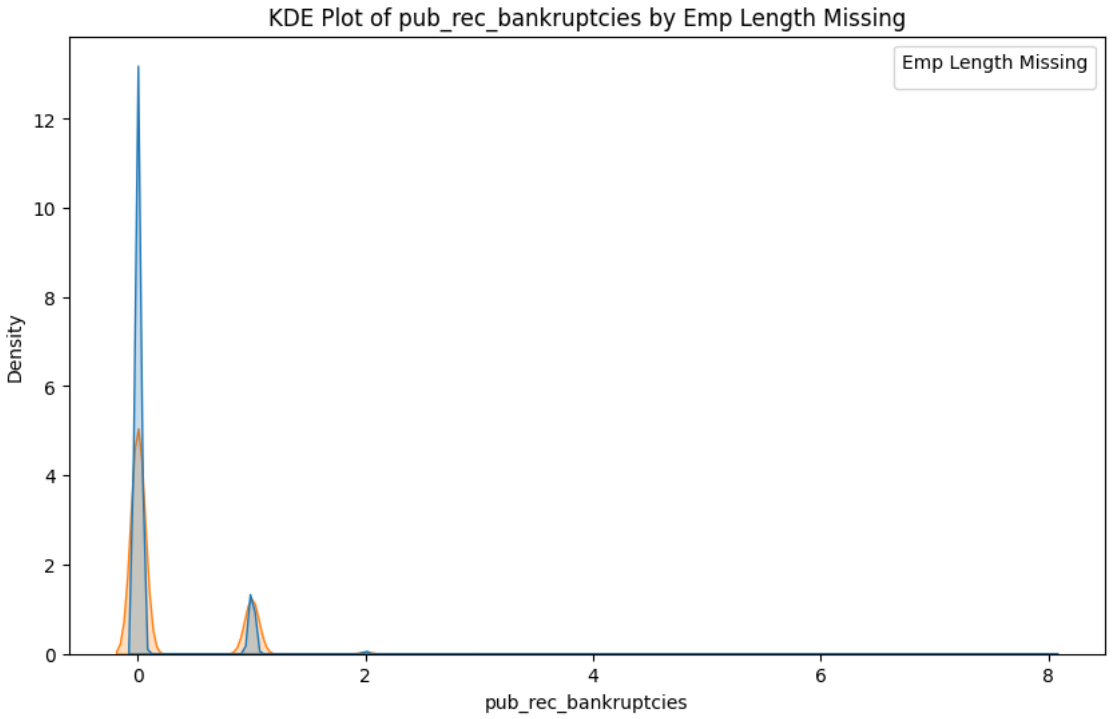

```
C:\Users\sreem\AppData\Local\Temp\ipykernel_34316\3191714987.py:28: UserWarning:
No artists with labels found to put in legend.  Note that artists whose label
start with an underscore are ignored when legend() is called with no argument.
  plt.legend(title='Emp Length Missing')
```

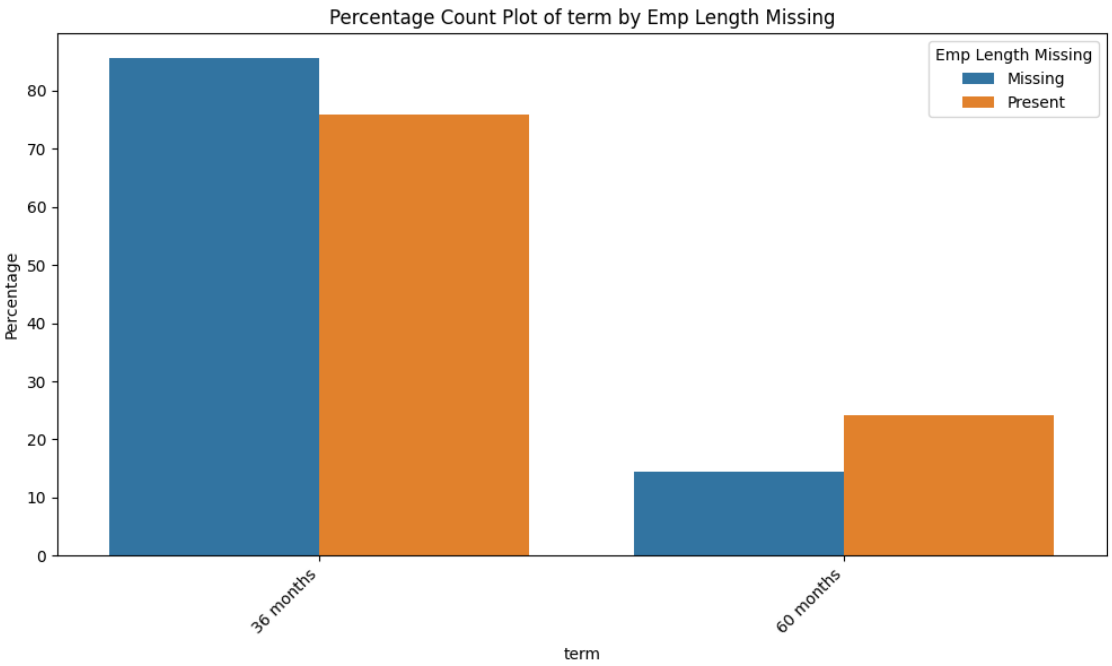


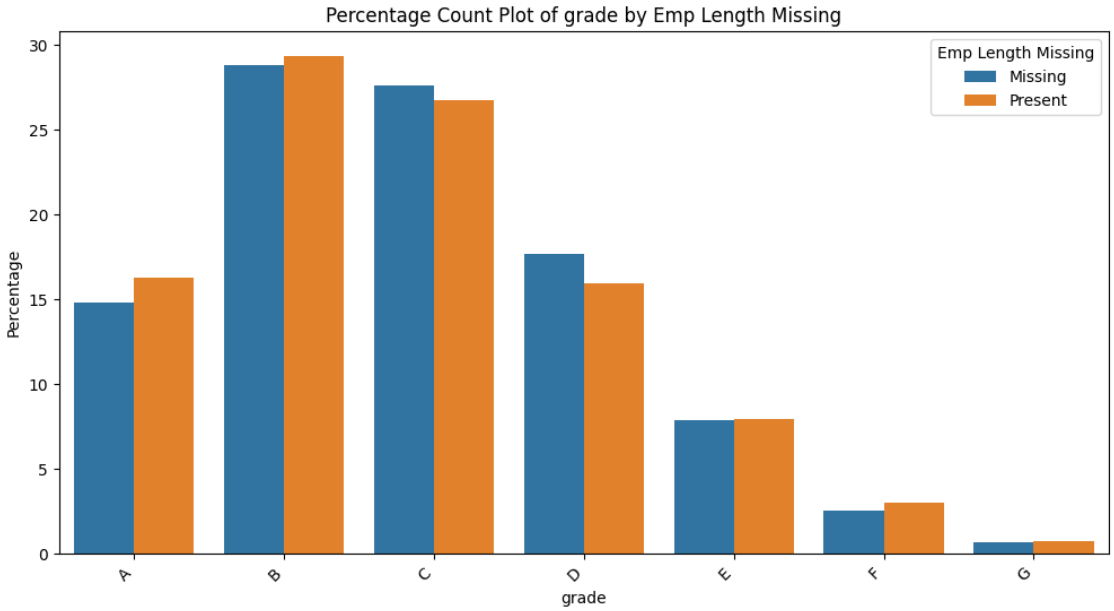
```
C:\Users\sreem\AppData\Local\Temp\ipykernel_34316\3191714987.py:28: UserWarning:
No artists with labels found to put in legend.  Note that artists whose label
start with an underscore are ignored when legend() is called with no argument.
  plt.legend(title='Emp Length Missing')
```

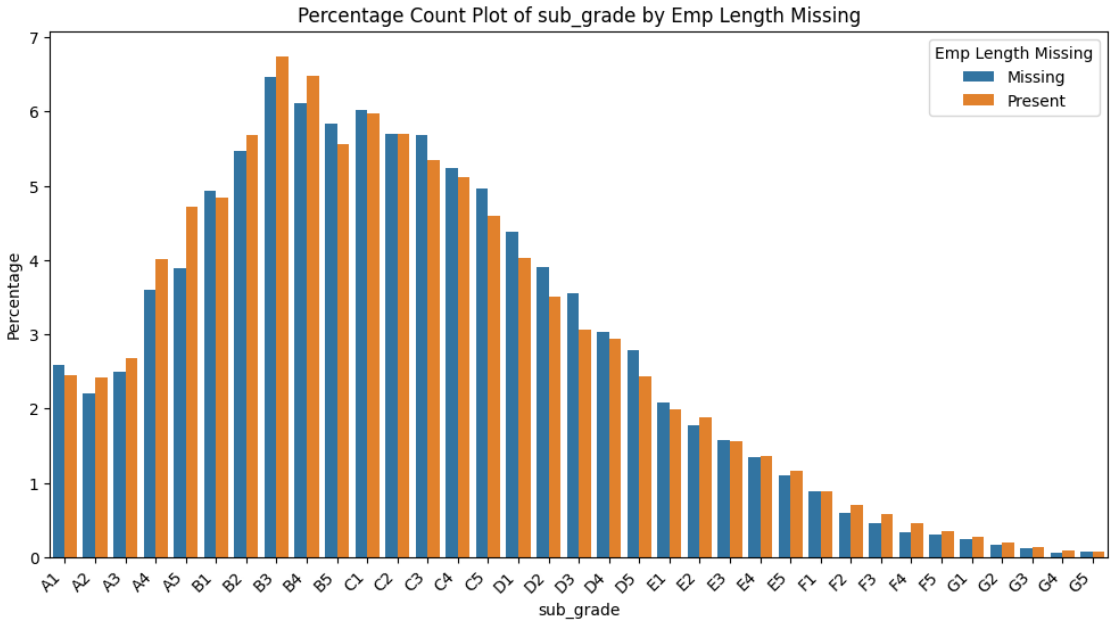


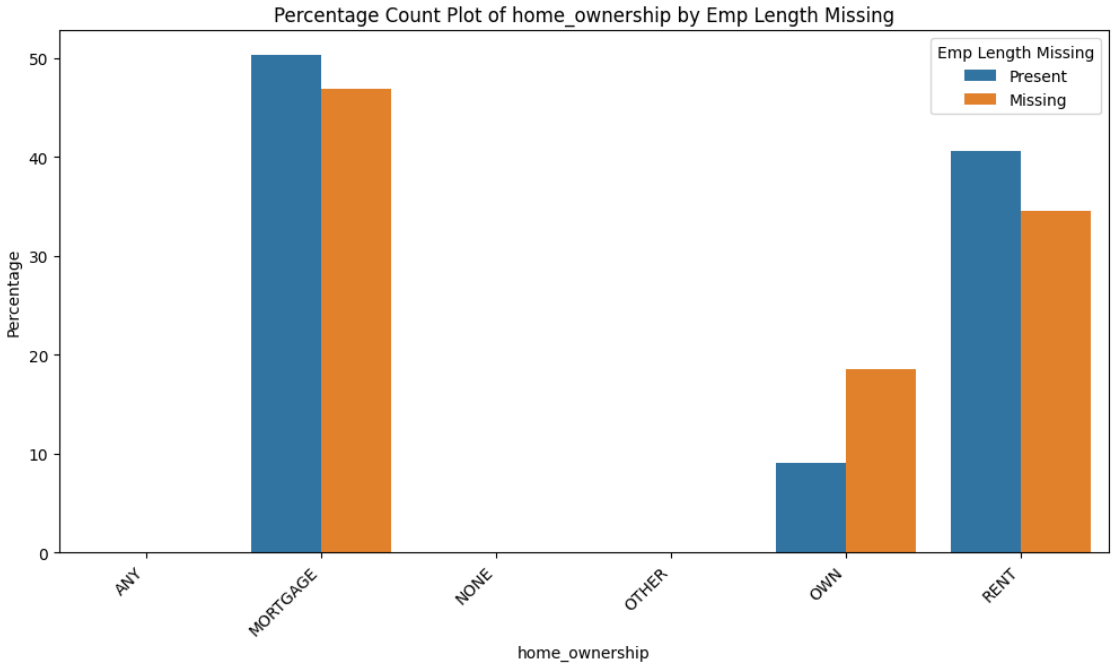

```
C:\Users\sreem\AppData\Local\Temp\ipykernel_34316\3191714987.py:28: UserWarning:
No artists with labels found to put in legend.  Note that artists whose label
start with an underscore are ignored when legend() is called with no argument.
  plt.legend(title='Emp Length Missing')
```

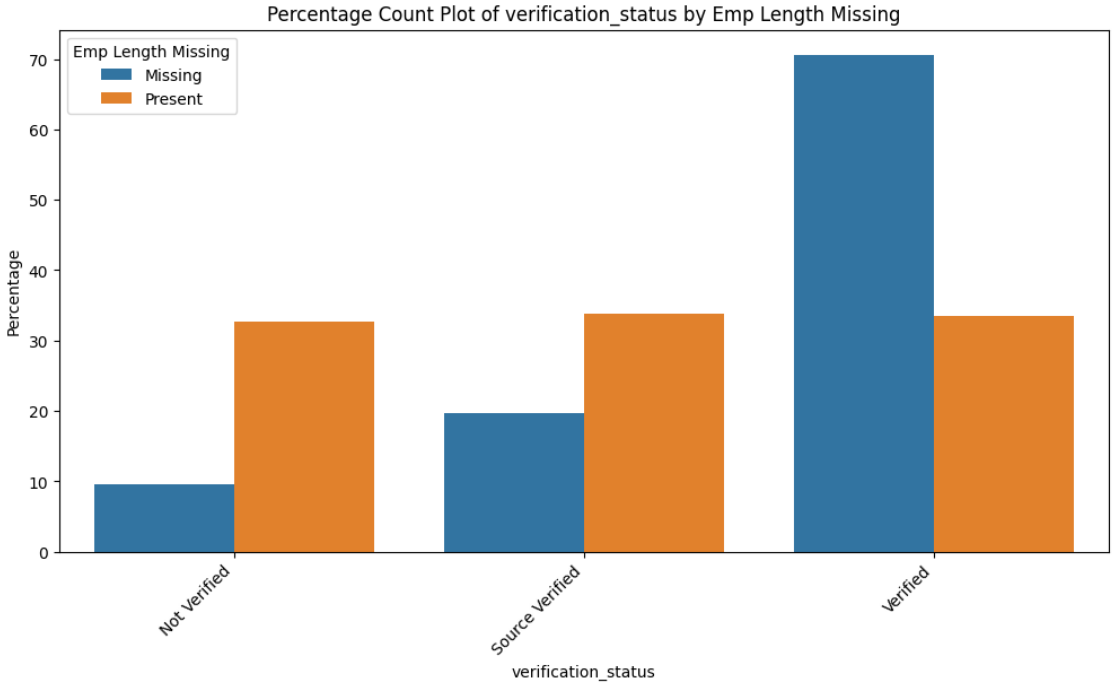

KDE Plot of pub_rec_bankruptcies by Emp Length Missing
Emp Length Missing
Density
pub_rec_bankruptcies
0
2
4
6
8
10
12

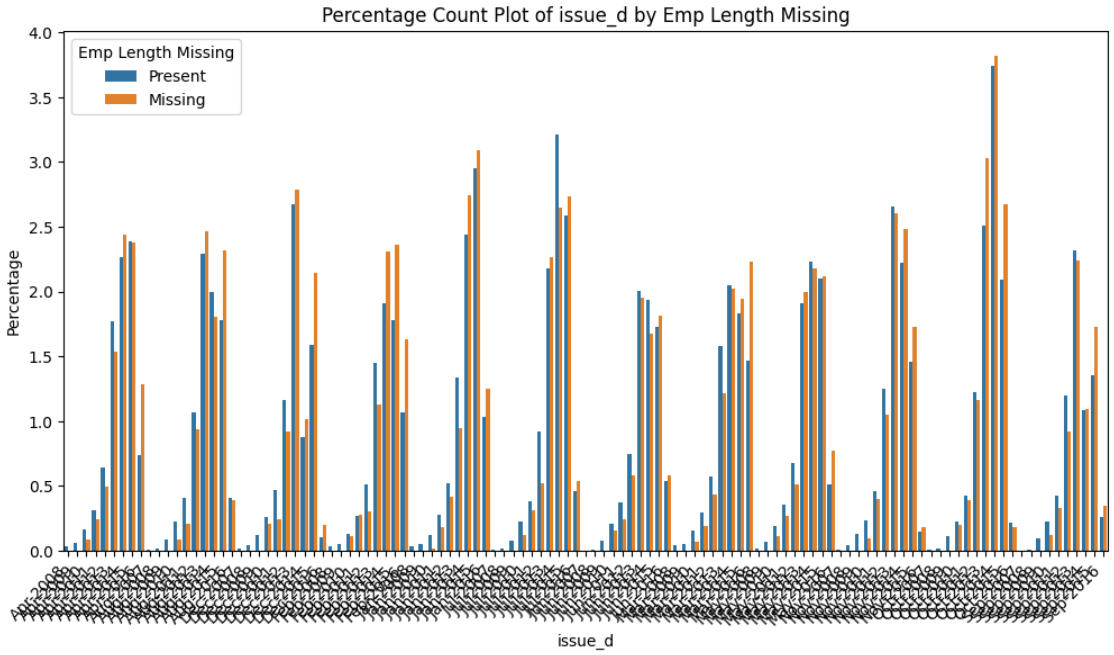

Percentage Count Plot of term by Emp Length Missing
Emp Length Missing
Missing
Present
Percentage
term
36 months
60 months
0
10
20
30
40
50
60
70
80

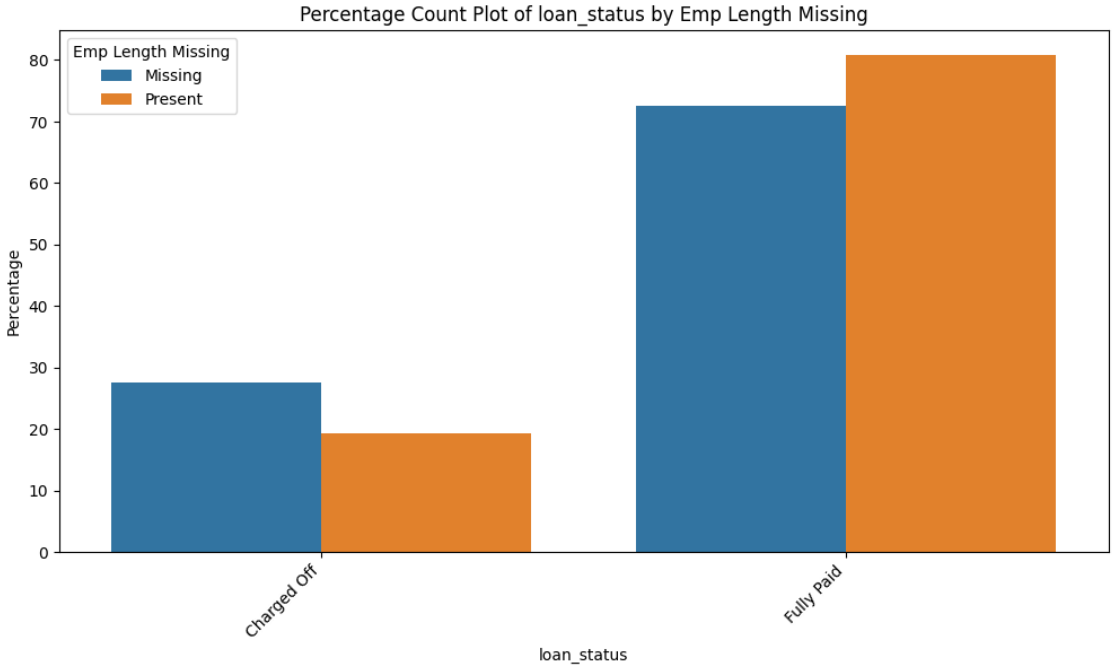

Percentage Count Plot of grade by Emp Length Missing
Emp Length Missing
Missing
Present
Percentage
grade
A
B
C
D
E
F
G

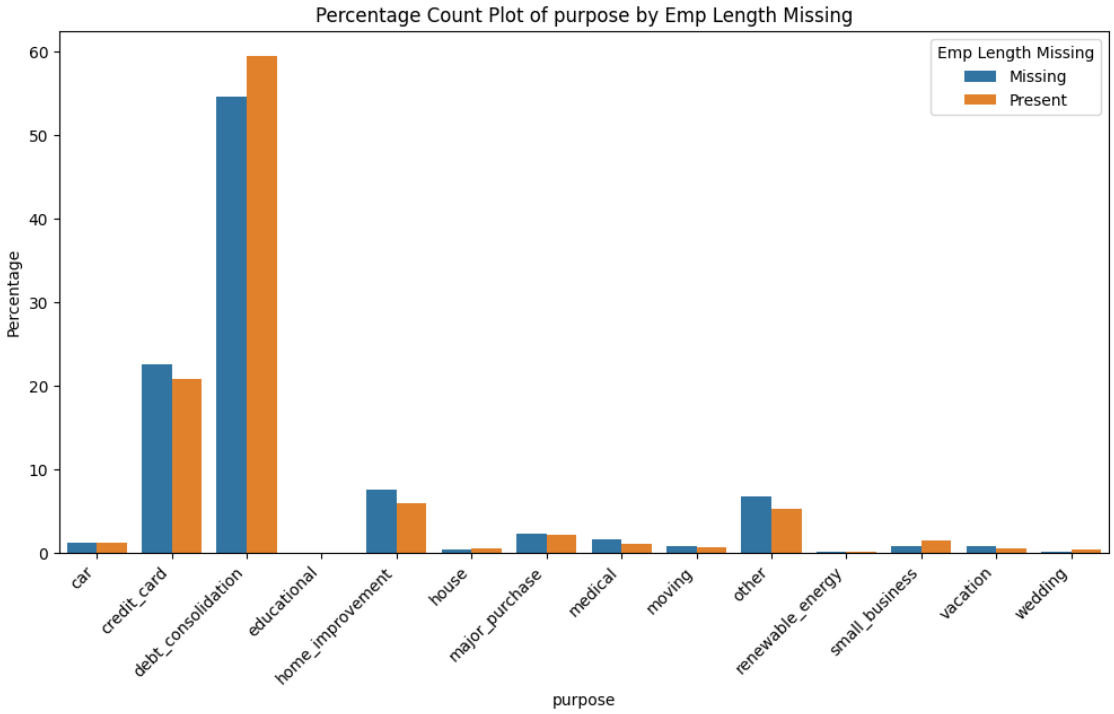

Percentage Count Plot of sub_grade by Emp Length Missing
Emp Length Missing
Missing
Present
Percentage
sub_grade

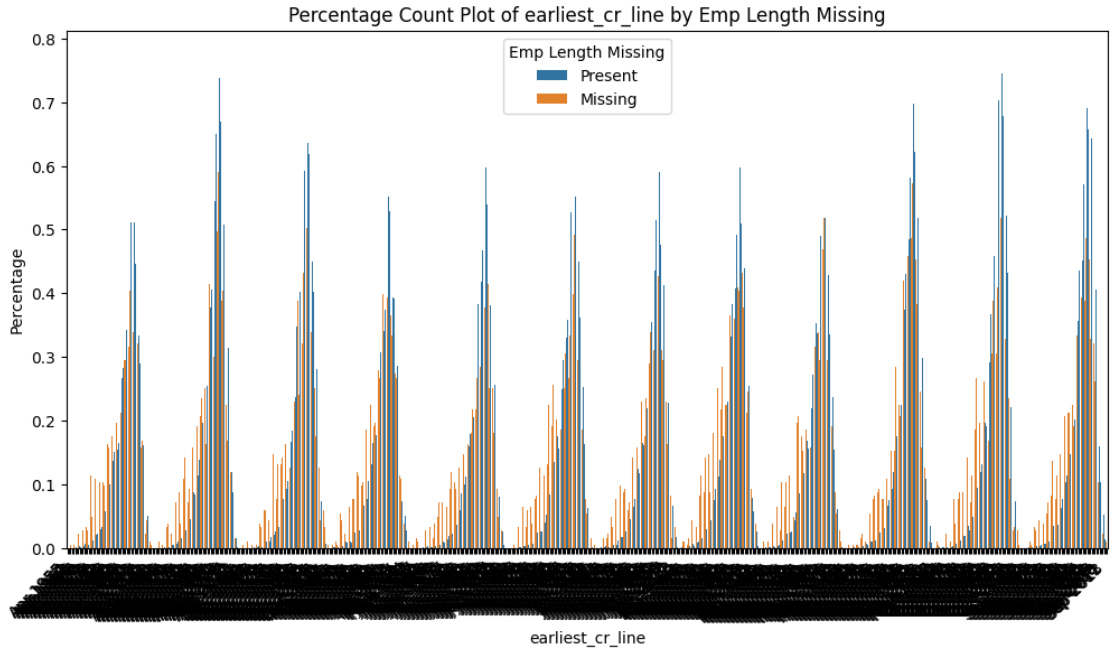

Percentage Count Plot of home_ownership by Emp Length Missing
Emp Length Missing
Present
Missing
Percentage
0
10
20
30
40
50
ANY
MORTGAGE
NONE
OTHER
OWN
RENT
home_ownership

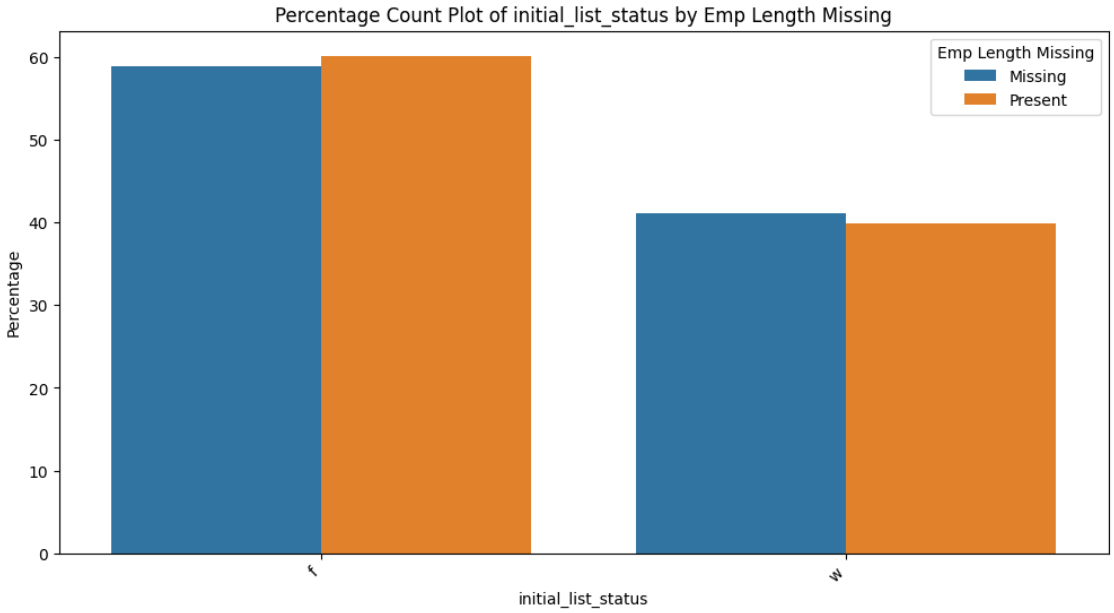

Percentage Count Plot of verification_status by Emp Length Missing
Emp Length Missing
Missing
Present
Percentage
0
10
20
30
40
50
60
70
Not Verified
Source Verified
Verified
verification_status

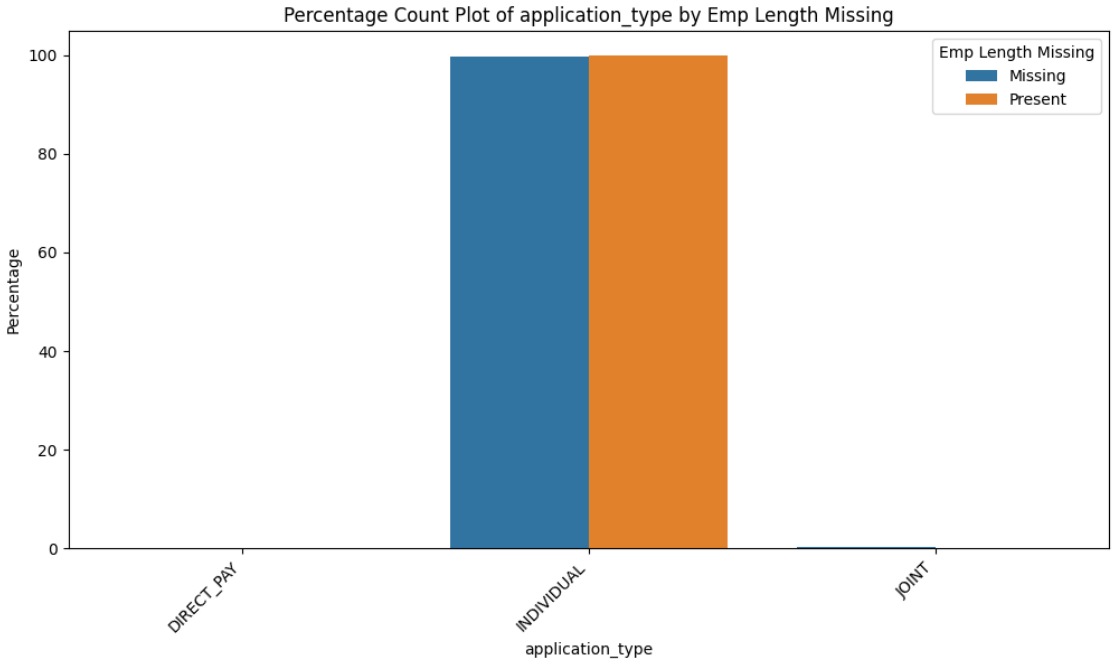

Percentage Count Plot of issue_d by Emp Length Missing
Emp Length Missing
Present
Missing
Percentage
issue_d


Percentage Count Plot of loan_status by Emp Length Missing
Emp Length Missing
Missing
Present
Percentage
Charged Off
Fully Paid
loan_status


Percentage Count Plot of purpose by Emp Length Missing
Emp Length Missing
Missing
Present
Percentage
0
10
20
30
40
50
60
car
credit_card
debt_consolidation
educational
home_improvement
house
major_purchase
medical
moving
other
renewable_energy
small_business
vacation
wedding
purpose


Percentage Count Plot of earliest_cr_line by Emp Length Missing
Emp Length Missing
Present
Missing
Percentage
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
earliest_cr_line





Observations: - There is no significant relation between `emp_lenght` and other features, since above plots are the same before and after.

5.0.2 Data Imputation

[ ]:
```
loan_data['emp_length'] = loan_data['emp_length'].fillna('< 1 year')
```

[ ]:
```
loan_data.loc[loan_data['revol_util'].isna(),'revol_util'] = 0.0
loan_data.loc[loan_data['mort_acc'].isna(),'mort_acc'] = 0.0
loan_data.loc[loan_data['pub_rec_bankruptcies'].isna(),'pub_rec_bankruptcies']␣
↪= 0.0
```

Checking Null values again after Imputation.

[ ]:
```
sum(loan_data.isna().sum())
```

[ ]:
```
0
```

[ ]:
```
loan_data.head()
```

[ ]:
```
   loan_amnt        term  int_rate  installment grade sub_grade  \
0    10000.0   36 months     11.44       329.48     B        B4
1     8000.0   36 months     11.99       265.68     B        B5
2    15600.0   36 months     10.49       506.97     B        B3
3     7200.0   36 months      6.49       220.65     A        A2
4    24375.0   60 months     17.27       609.33     C        C5

                emp_title emp_length home_ownership  annual_inc  \
0               Marketing  10+ years           RENT    117000.0
1         Credit analyst     4 years       MORTGAGE     65000.0
2            Statistician   < 1 year           RENT     43057.0
3         Client Advocate    6 years           RENT     54000.0
4  Destiny Management Inc.    9 years       MORTGAGE     55000.0

  verification_status   issue_d  loan_status             purpose  \
0       Not Verified  Jan-2015   Fully Paid            vacation
1       Not Verified  Jan-2015   Fully Paid  debt_consolidation
2    Source Verified  Jan-2015   Fully Paid         credit_card
3       Not Verified  Nov-2014   Fully Paid         credit_card
4           Verified  Apr-2013  Charged Off         credit_card

                   title    dti earliest_cr_line  open_acc  pub_rec  \
0                Vacation  26.24         Jun-1990      16.0      0.0
1      Debt consolidation  22.05         Jul-2004      17.0      0.0
2  Credit card refinancing  12.79         Aug-2007      13.0      0.0
3  Credit card refinancing   2.60         Sep-2006       6.0      0.0
4    Credit Card Refinance  33.95         Mar-1999      13.0      0.0

   revol_bal  revol_util  total_acc initial_list_status application_type  \
0    36369.0        41.8       25.0                   w       INDIVIDUAL
1    20131.0        53.3       27.0                   f       INDIVIDUAL
```

```
2    11987.0        92.2       26.0                  f       INDIVIDUAL
3     5472.0        21.5       13.0                  f       INDIVIDUAL
4    24584.0        69.8       43.0                  f       INDIVIDUAL

   mort_acc  pub_rec_bankruptcies  \
0       0.0                   0.0
1       3.0                   0.0
2       0.0                   0.0
3       0.0                   0.0
4       1.0                   0.0

                                              address emp_length_missing
0     0174 Michelle Gateway\r\nMendozaberg, OK 22690            Present
1  1076 Carney Fort Apt. 347\r\nLoganmouth, SD 05113            Present
2  87025 Mark Dale Apt. 269\r\nNew Sabrina, WV 05113            Present
3            823 Reid Ford\r\nDelacruzside, MA 00813            Present
4             679 Luna Roads\r\nGreggshire, VA 11650            Present
```

6 Feature Engineering

[ ]:
```
loan_data['pub_rec'] = [1 if i > 1 else 0 for i in loan_data['pub_rec']]
loan_data['mort_acc'] = [1 if i > 1 else 0 for i in loan_data['mort_acc']]
loan_data['pub_rec_bankruptcies'] = [1 if i > 1 else 0 for i in␣
 ↪loan_data['pub_rec_bankruptcies']]
```

[ ]:
```
#Split issue_date into month and year
loan_data[['issue_month', 'issue_year']] = loan_data['issue_d'].str.split('-',␣
 ↪expand=True)
```

[ ]:
```
#Split er_cr_line date into month and year
loan_data[['er_cr_line_m', 'er_cr_line_y']] = loan_data['earliest_cr_line'].str.
 ↪split('-', expand=True)
```

[ ]:
```
# Conversion of issue_d and earliest_cr_line columns to datetime datatype
loan_data["issue_d"] = loan_data["issue_d"].astype('datetime64[ns]')
loan_data["earliest_cr_line"] = loan_data["earliest_cr_line"].
 ↪astype('datetime64[ns]')
loan_data["days_from_first_loan"] = ␣
 ↪(loan_data["issue_d"]-loan_data["earliest_cr_line"]).dt.days
```

[ ]:
```
loan_data.drop(['earliest_cr_line'], axis=1, inplace=True)
loan_data.drop(['issue_d'], axis=1, inplace=True)
```

[ ]:
```
#Split address into State and Zip code
loan_data[['state','zipcode']] = loan_data['address'].str.extract(r'([A-Z]{2})␣
 ↪(\d{5})')
```

```
loan_data.drop(['address'], axis=1, inplace=True)
```

[ ]:
```
state_dict = {"AA":"Armed Forces America", "AE":"Armed Forces","AK":
 ↪"Alaska","AL":"Alabama",
              "AP":"Armed Forces Pacific","AR":"Arkansas","AZ":"Arizona","CA":
 ↪"California",
              "CO":"Colorodo","CT":"Connecticut","DC":"Washington DC","DE":
 ↪"Delaware",
              "FL":"Florida","GA":"Georgia","HI":"Hawaii","IA":"Iowa","ID":
 ↪"Idaho","IL":"Illinois",
              "IN":"Indiana","KS":"Kansas","KY":"Kentucky","LA":
 ↪"Louisiana","MA":"Massachusetts",
              "MD":"Maryland","ME":"Maine","MI":"Michigan","MN":
 ↪"Minnesota","MO":"Missouri",
              "MS":"Mississippi","MT":"Montana","NC":"North Carolina","ND":
 ↪"North Dakota",
              "NE":"Nebraska","NH":"New Hampshire","NJ":"New Jersey","NM":"New␣
 ↪Mexico","NV":"Nevada",
              "NY":"New York","OH":"Ohio","OK":"Oklahoma","OR":"Oregon","PA":
 ↪"Pennsylvania",
              "RI":"Rhode Island","SC":"South Carolina","SD":"South␣
 ↪Dakota","TN":"Tennessee",
              "TX":"Texas","UT":"Utah","VA":"Virginia","VT":"Vermont","WA":
 ↪"Washington",
              "WI":"Wisconsin","WV":"West Virginia","WY":"Wyoming"}
```

[ ]:
```
loan_data["state"] = loan_data.state.map(state_dict)
```

[ ]:
```
loan_data['term'] = loan_data['term'].str.split().str[0].astype(int)
```

[ ]:
```
loan_data['emp_length_yrs'] =loan_data['emp_length'].str.extract('(\d+)').
 ↪astype(int)
```

[ ]:
```
loan_data.drop(['emp_length'], axis=1, inplace=True)
```

[ ]:
```
loan_data['emp_length_yrs'].value_counts()
```

[ ]:
```
emp_length_yrs
10    126041
1      75908
2      35827
3      31665
5      26495
4      23952
6      20841
7      20819
```

```
8      19168
9      15314
Name: count, dtype: int64
```

[ ]:
```
loan_data.head()
```

[ ]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	\
0	10000.0	36	11.44	329.48	B	B4	
1	8000.0	36	11.99	265.68	B	B5	
2	15600.0	36	10.49	506.97	B	B3	
3	7200.0	36	6.49	220.65	A	A2	
4	24375.0	60	17.27	609.33	C	C5	

	emp_title	home_ownership	annual_inc	verification_status	\
0	Marketing	RENT	117000.0	Not Verified	
1	Credit analyst	MORTGAGE	65000.0	Not Verified	
2	Statistician	RENT	43057.0	Source Verified	
3	Client Advocate	RENT	54000.0	Not Verified	
4	Destiny Management Inc.	MORTGAGE	55000.0	Verified	

	loan_status	purpose	title	dti	open_acc	\
0	Fully Paid	vacation	Vacation	26.24	16.0	
1	Fully Paid	debt_consolidation	Debt consolidation	22.05	17.0	
2	Fully Paid	credit_card	Credit card refinancing	12.79	13.0	
3	Fully Paid	credit_card	Credit card refinancing	2.60	6.0	
4	Charged Off	credit_card	Credit Card Refinance	33.95	13.0	

	pub_rec	revol_bal	revol_util	total_acc	initial_list_status	\
0	0	36369.0	41.8	25.0	w	
1	0	20131.0	53.3	27.0	f	
2	0	11987.0	92.2	26.0	f	
3	0	5472.0	21.5	13.0	f	
4	0	24584.0	69.8	43.0	f	

	application_type	mort_acc	pub_rec_bankruptcies	emp_length_missing	\
0	INDIVIDUAL	0	0	Present	
1	INDIVIDUAL	1	0	Present	
2	INDIVIDUAL	0	0	Present	
3	INDIVIDUAL	0	0	Present	
4	INDIVIDUAL	0	0	Present	

	issue_month	issue_year	er_cr_line_m	er_cr_line_y	days_from_first_loan	\
0	Jan	2015	Jun	1990	8980	
1	Jan	2015	Jul	2004	3836	
2	Jan	2015	Aug	2007	2710	
3	Nov	2014	Sep	2006	2983	
4	Apr	2013	Mar	1999	5145	

```
            state zipcode  emp_length_yrs
0        Oklahoma   22690              10
1    South Dakota   05113               4
2   West Virginia   05113               1
3   Massachusetts   00813               6
4        Virginia   11650               9
```

Observations: - `pub_rec`, `mort_accounts`, `pub_rec_bankruptcies` are very important features for the prediction of **Defaulters**. Even if the values are greater than or equal to 1, the person is to be treated as a defaulter. Therefore these features are coverted to binary (1 = person with 1 or more bad records in these features) - Date and month are seperated from the datetime related columns[issue_d,earliest_cr_line]. No of days from `days_from_first_loan`. - State and Zipcode is extracted from address using regex. - `term`, `emp_lenght` extracted the numbers from these columns.

7 Data Visulaization

[ ]:
```
loan_data.dtypes
```

[ ]:
```
loan_amnt              float64
term                     int64
int_rate               float64
installment            float64
grade                   object
sub_grade               object
emp_title               object
home_ownership          object
annual_inc             float64
verification_status     object
loan_status             object
purpose                 object
title                   object
dti                    float64
open_acc               float64
pub_rec                  int64
revol_bal              float64
revol_util             float64
total_acc              float64
initial_list_status     object
application_type        object
mort_acc                 int64
pub_rec_bankruptcies     int64
emp_length_missing      object
issue_month             object
issue_year              object
er_cr_line_m            object
```

```
er_cr_line_y             object
days_from_first_loan      int64
state                    object
zipcode                  object
emp_length_yrs            int64
dtype: object
```

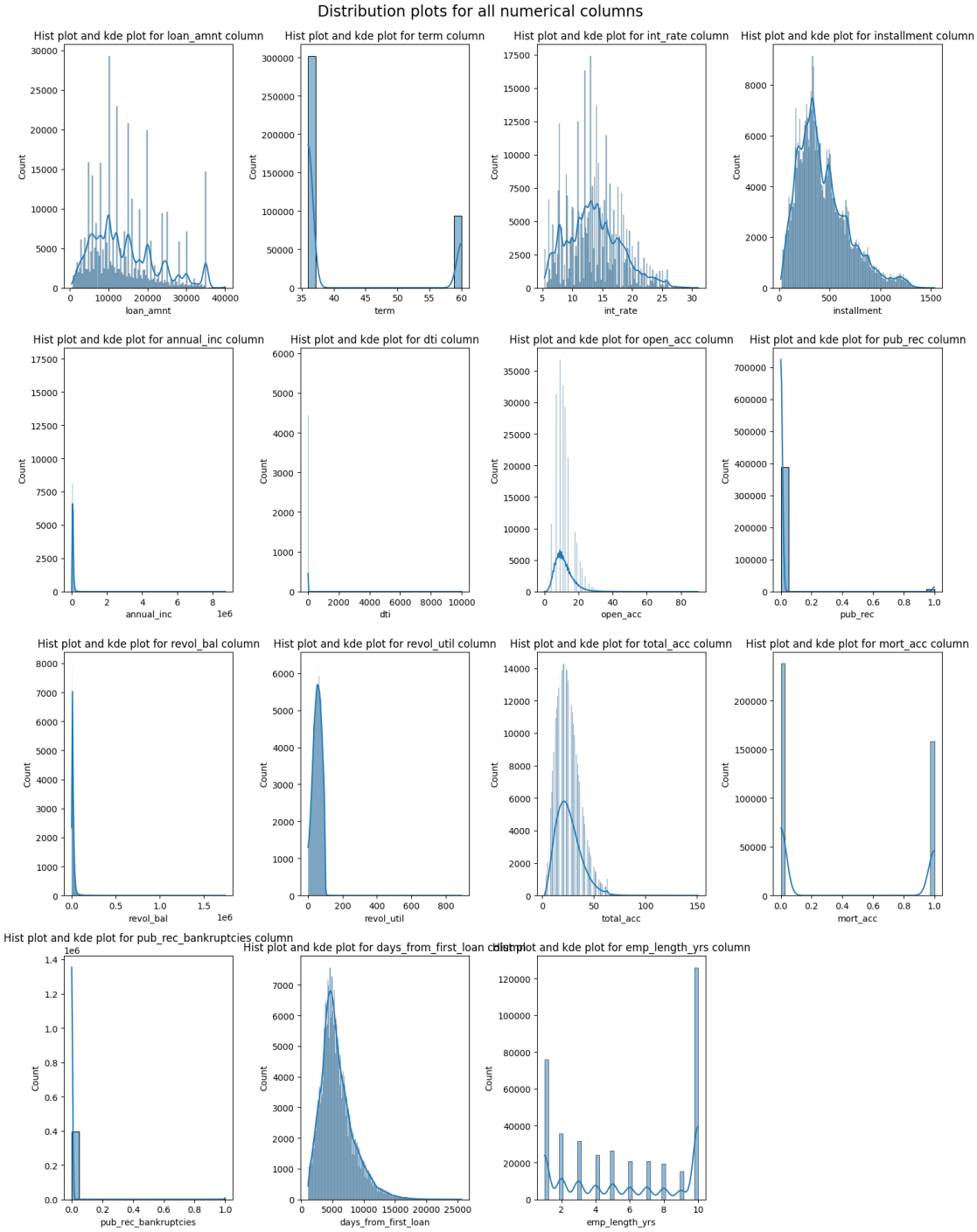
[ ]:
```
numerical_cols = loan_data.select_dtypes(include=['float64', 'int64']).columns.
  ↪tolist()
categorical_cols = loan_data.select_dtypes(include=['object']).columns.tolist()
print(numerical_cols)
print(categorical_cols)
```

```
['loan_amnt', 'term', 'int_rate', 'installment', 'annual_inc', 'dti',
'open_acc', 'pub_rec', 'revol_bal', 'revol_util', 'total_acc', 'mort_acc',
'pub_rec_bankruptcies', 'days_from_first_loan', 'emp_length_yrs']
['grade', 'sub_grade', 'emp_title', 'home_ownership', 'verification_status',
'loan_status', 'purpose', 'title', 'initial_list_status', 'application_type',
'emp_length_missing', 'issue_month', 'issue_year', 'er_cr_line_m',
'er_cr_line_y', 'state', 'zipcode']
```

7.1 Univariate

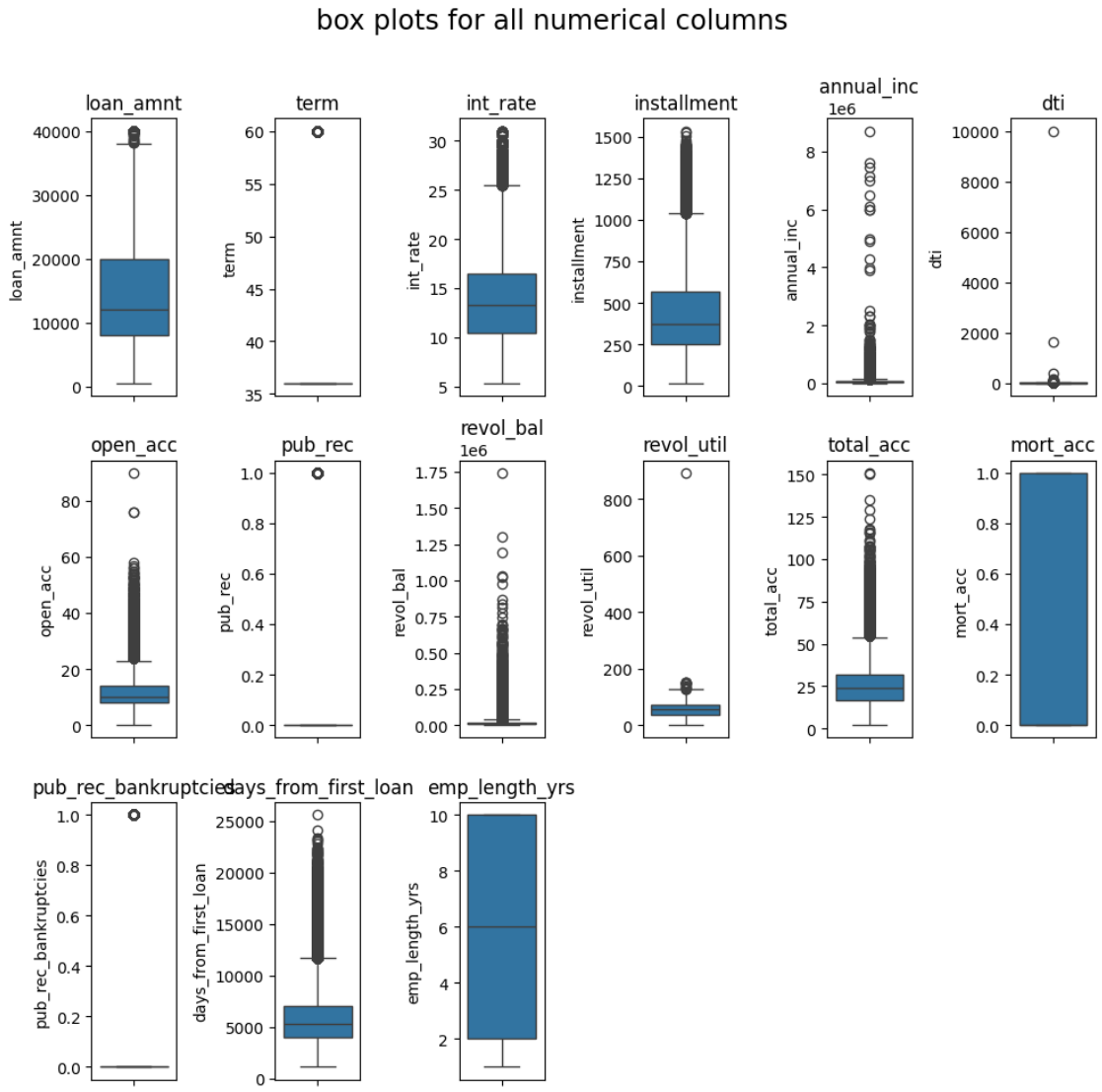
7.1.1 Distribution of all numerical cols

[ ]:
```
fig = plt.figure(figsize = (15,20))
fig.suptitle("Distribution plots for all numerical columns\n",fontsize =␣
  ↪"xx-large" )
k = 1
for i in numerical_cols:
    plt.subplot(4,4,k)
    plt.title("Hist plot and kde plot for {} column".format(i))
    sns.histplot(data=loan_data, x=i, kde=True)
    k = k+1
plt.tight_layout()
plt.show()
```


Distribution plots for all numerical columns
Hist plot and kde plot for loan_amnt column
loan_amnt
Hist plot and kde plot for term column
term
Hist plot and kde plot for int_rate column
int_rate
Hist plot and kde plot for installment column
installment
Hist plot and kde plot for annual_inc column
annual_inc
1e6
Hist plot and kde plot for dti column
dti
Hist plot and kde plot for open_acc column
open_acc
Hist plot and kde plot for pub_rec column
pub_rec
Hist plot and kde plot for revol_bal column
revol_bal
1e6
Hist plot and kde plot for revol_util column
revol_util
Hist plot and kde plot for total_acc column
total_acc
Hist plot and kde plot for mort_acc column
mort_acc
Hist plot and kde plot for pub_rec_bankruptcies column
1e6
pub_rec_bankruptcies
Hist plot and kde plot for days_from_first_loan column
days_from_first_loan
Hist plot and kde plot for emp_length_yrs column
emp_length_yrs
Count

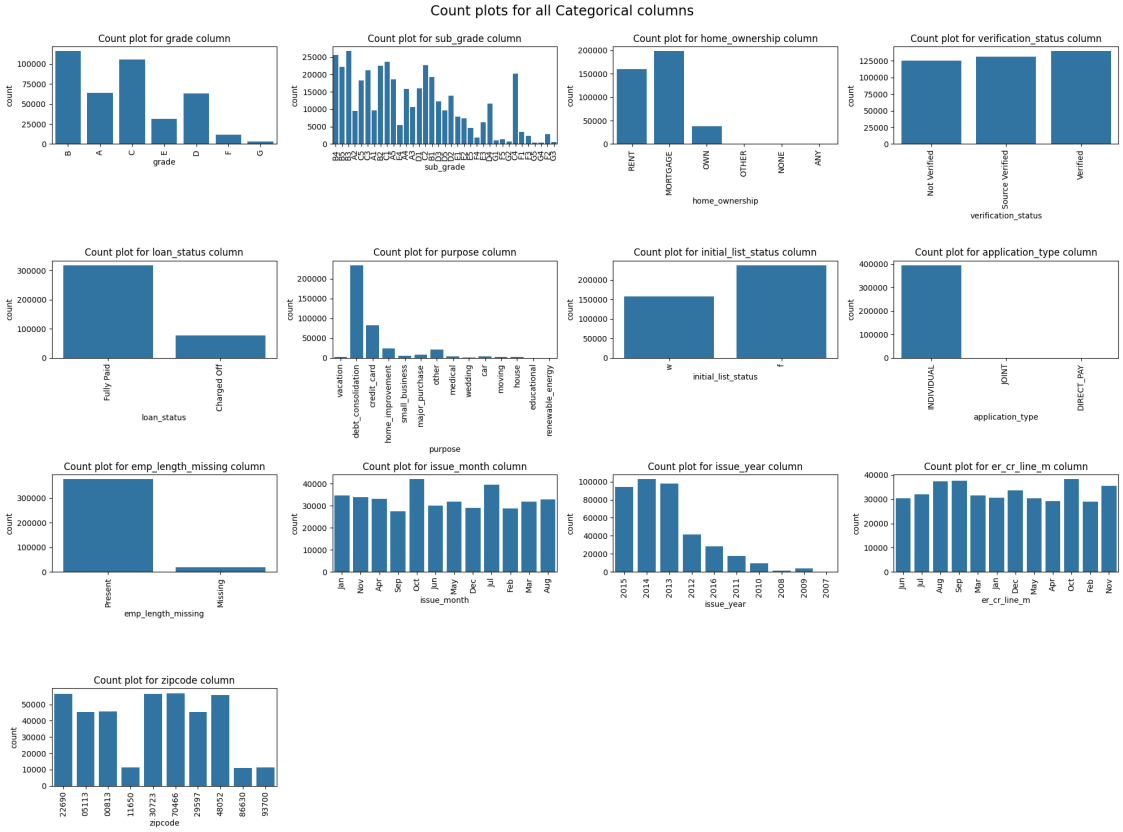
7.1.2 Box plot of all numerical cols

```
[ ]: fig = plt.figure(figsize = (10,10))
fig.suptitle("box plots for all numerical columns\n",fontsize = "xx-large" )
k = 1
for i in numerical_cols:
    plt.subplot(3,6,k)
    plt.title("{}".format(i))
    sns.boxplot(data=loan_data[i])
    k = k+1
plt.tight_layout()
plt.show()
```



7.1.3 Count Plots of all categorical columns

```
[ ]: fig = plt.figure(figsize = (20,15))
fig.suptitle("Count plots for all Categorical columns\n",fontsize = "xx-large" )
k = 1
for i in categorical_cols:
    if loan_data[i].nunique()<50:
        plt.subplot(4,4,k)
        plt.title("Count plot for {} column".format(i))
        sns.countplot(data=loan_data, x=i)
        plt.xticks(rotation = 90)
        k = k+1
plt.tight_layout()
plt.show()
```

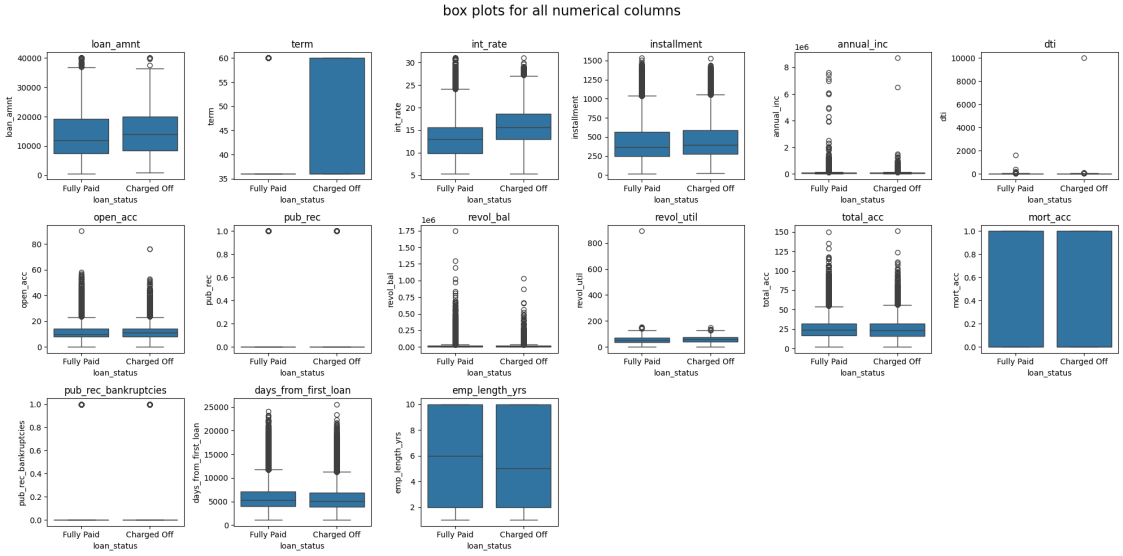


Observations: - Most numerical features such as annual_inc, dti, revol_bal, and days_from_first_loan are highly right-skewed with significant outliers. - Categorical variables like loan_status and application_type show heavy imbalance, suggesting the importance of sampling. - Employment length (emp_length_yrs) exhibits a sharp spike at 10 years and also has many entries around 0–1 years, with missing values needing thoughtful imputation since employment stability may influence loan risk. - The issue year shows most loans concentrated between 2015–2016, and while issue months are fairly uniformly distributed, creating time-based features

like loan vintage or seasonality indicators could enhance model performance.

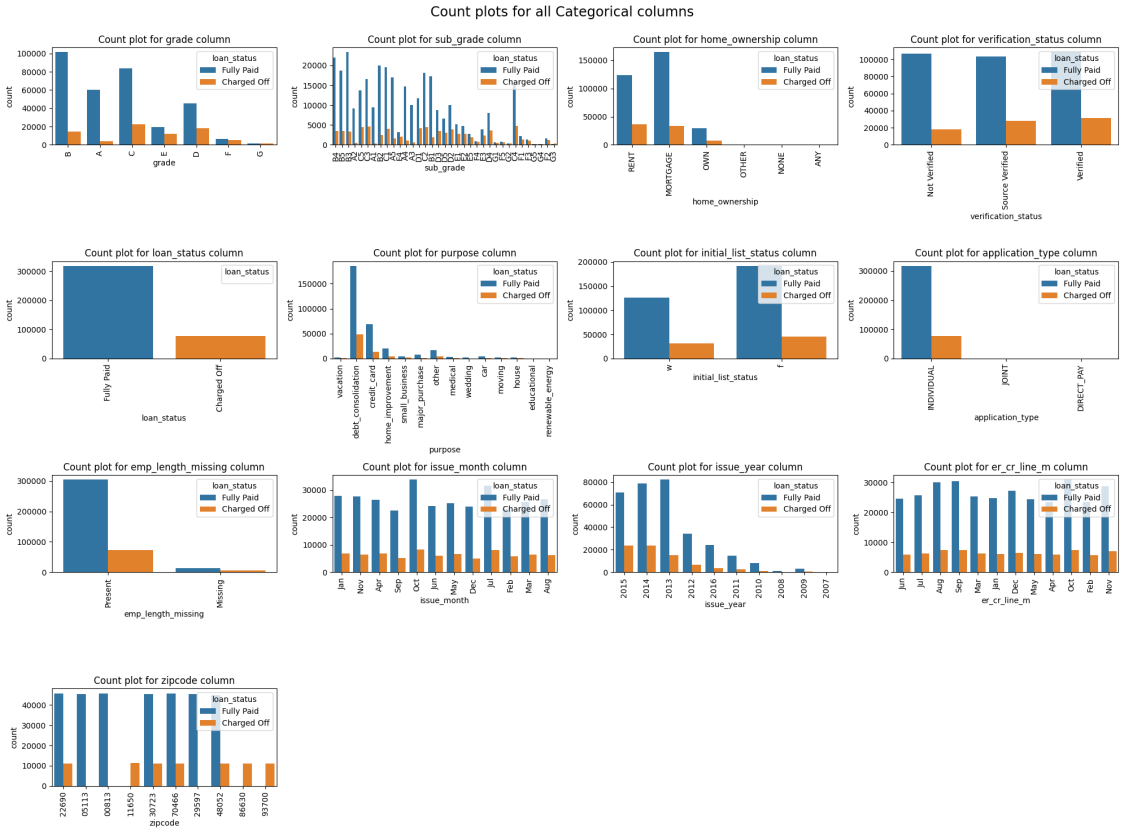
7.2 Bivariate

[ ]:
```
fig = plt.figure(figsize = (20,10))
fig.suptitle("box plots for all numerical columns\n",fontsize = "xx-large" )
k = 1
for i in numerical_cols:
    plt.subplot(3,6,k)
    plt.title("{}".format(i))
    sns.boxplot(data=loan_data,x = 'loan_status',y = i)
    k = k+1
plt.tight_layout()
plt.show()
```

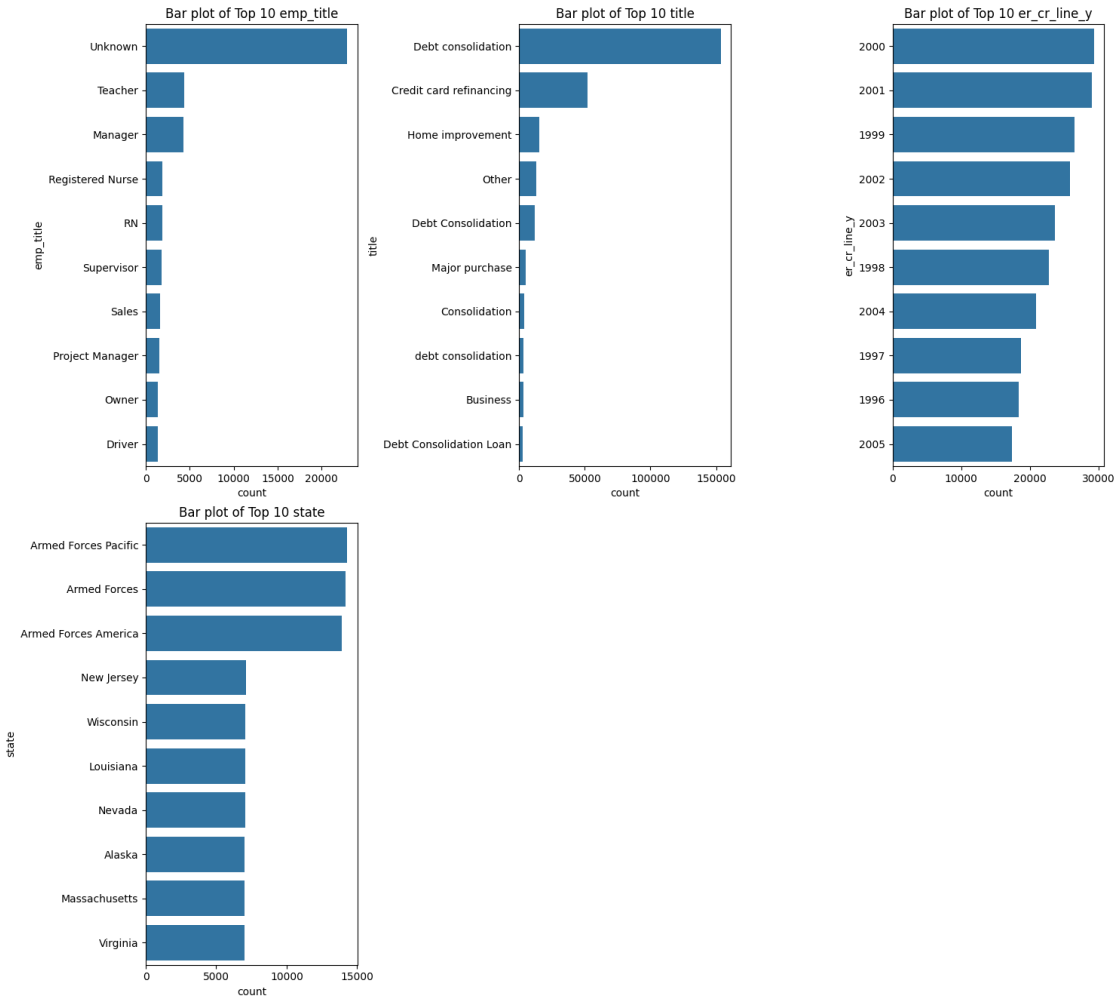


[ ]:
```
fig = plt.figure(figsize = (20,15))
fig.suptitle("Count plots for all Categorical columns\n",fontsize = "xx-large" )
k = 1
for i in categorical_cols:
    if loan_data[i].nunique()<50:
        plt.subplot(4,4,k)
        plt.title("Count plot for {} column".format(i))
        sns.countplot(data=loan_data, x=i, hue = 'loan_status')
        plt.legend(title = 'loan_status')
        plt.xticks(rotation = 90)
        k = k+1
plt.tight_layout()
plt.show()
```

```
C:\Users\sreem\AppData\Local\Temp\ipykernel_34316\2846516677.py:9: UserWarning:
No artists with labels found to put in legend.  Note that artists whose label
start with an underscore are ignored when legend() is called with no argument.
  plt.legend(title = 'loan_status')
```



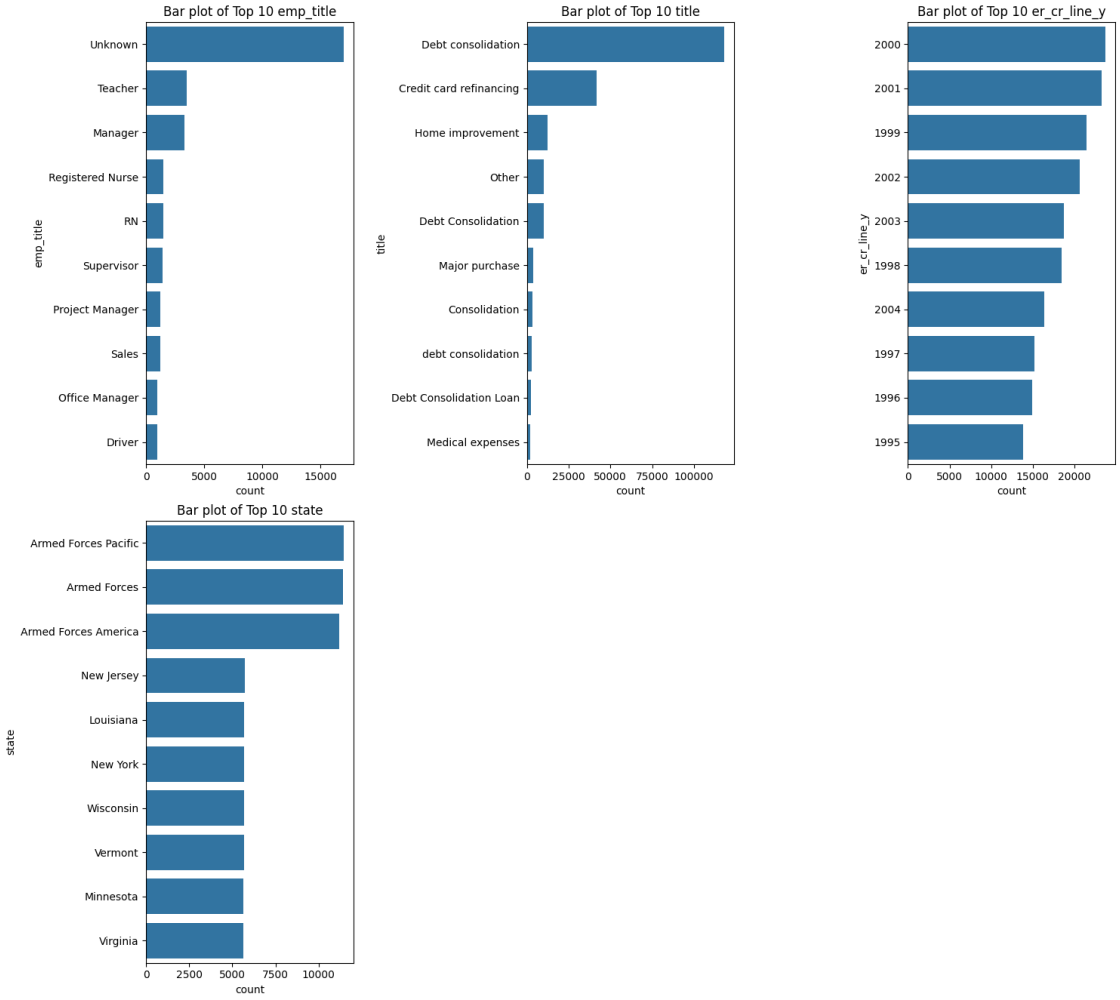
[ ]:
```
fig = plt.figure(figsize = (15, 15))
plt.suptitle("Bar plot of Top 10 values in Large Categorical␣
  ↪Columns\n",fontsize = "xx-large")
k = 1
for p in categorical_cols:
    if loan_data[p].nunique()>=50:
        plt.subplot(2,3,k)
        plt.title(f"Bar plot of Top 10 {p}")
        k += 1
        top_10 = loan_data[p].value_counts()[:10].reset_index()
        top_10.columns = [p, 'count']
        sns.barplot(x='count', y=p, data=top_10, orient='h')
plt.tight_layout()
plt.subplots_adjust(top=0.87)
plt.show()
```



[ ]:
```
fig = plt.figure(figsize = (15, 15))
plt.suptitle("Bar plot of Top 10 Fully Paid values in Large Categorical␣
 ↪Columns\n",fontsize = "xx-large")
k = 1
for p in categorical_cols:
    if loan_data[p].nunique()>=50:
        plt.subplot(2,3,k)
        plt.title(f"Bar plot of Top 10 {p}")
        k += 1
        top_10 = loan_data[loan_data['loan_status'] == 'Fully Paid'][p].
 ↪value_counts()[:10].reset_index()
        top_10.columns = [p, 'count']
        sns.barplot(x='count', y=p, data=top_10, orient='h')
```

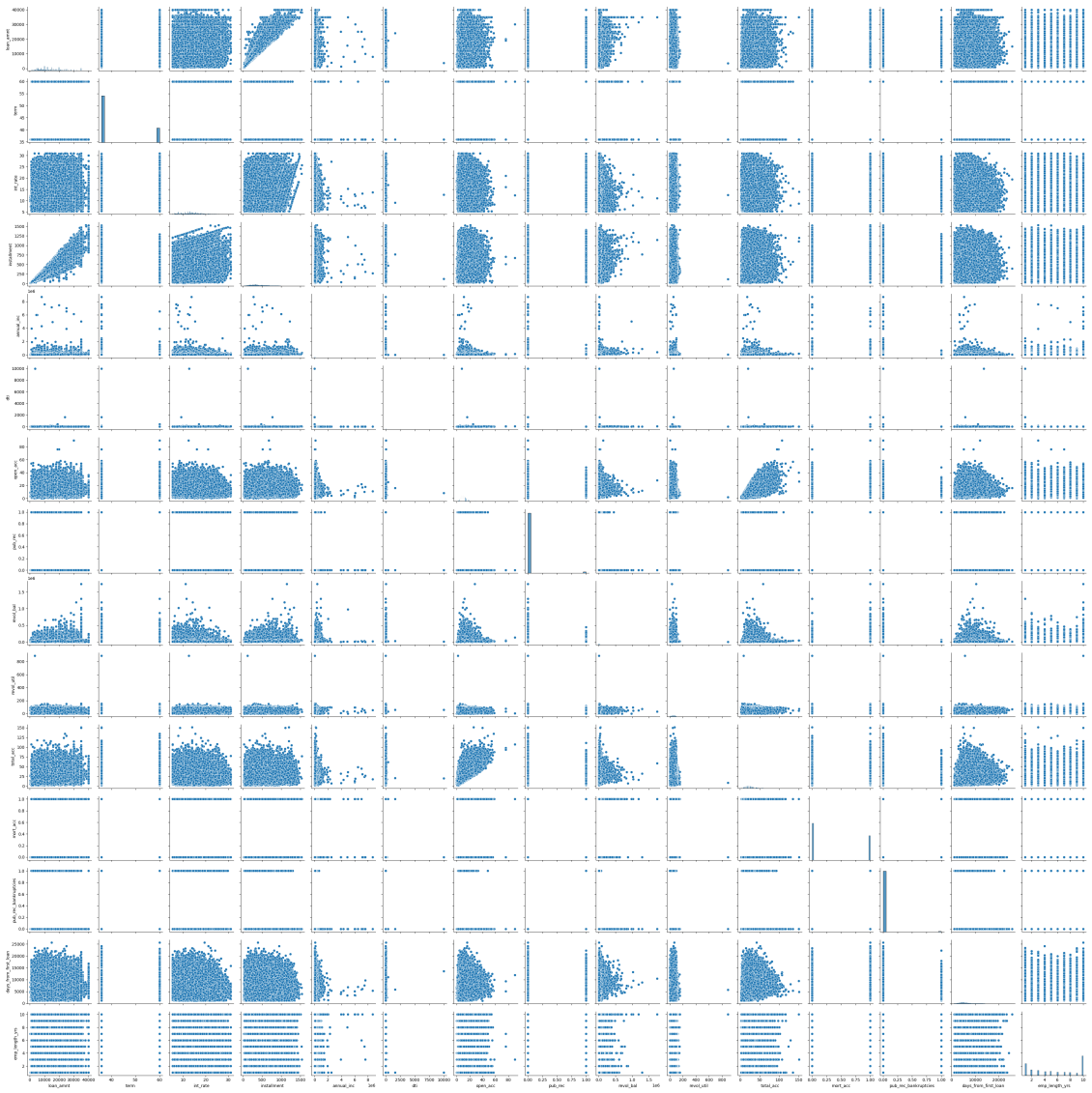
```
plt.tight_layout()
plt.subplots_adjust(top=0.87)
plt.show()
```

Bar plot of Top 10 Fully Paid values in Large Categorical Columns



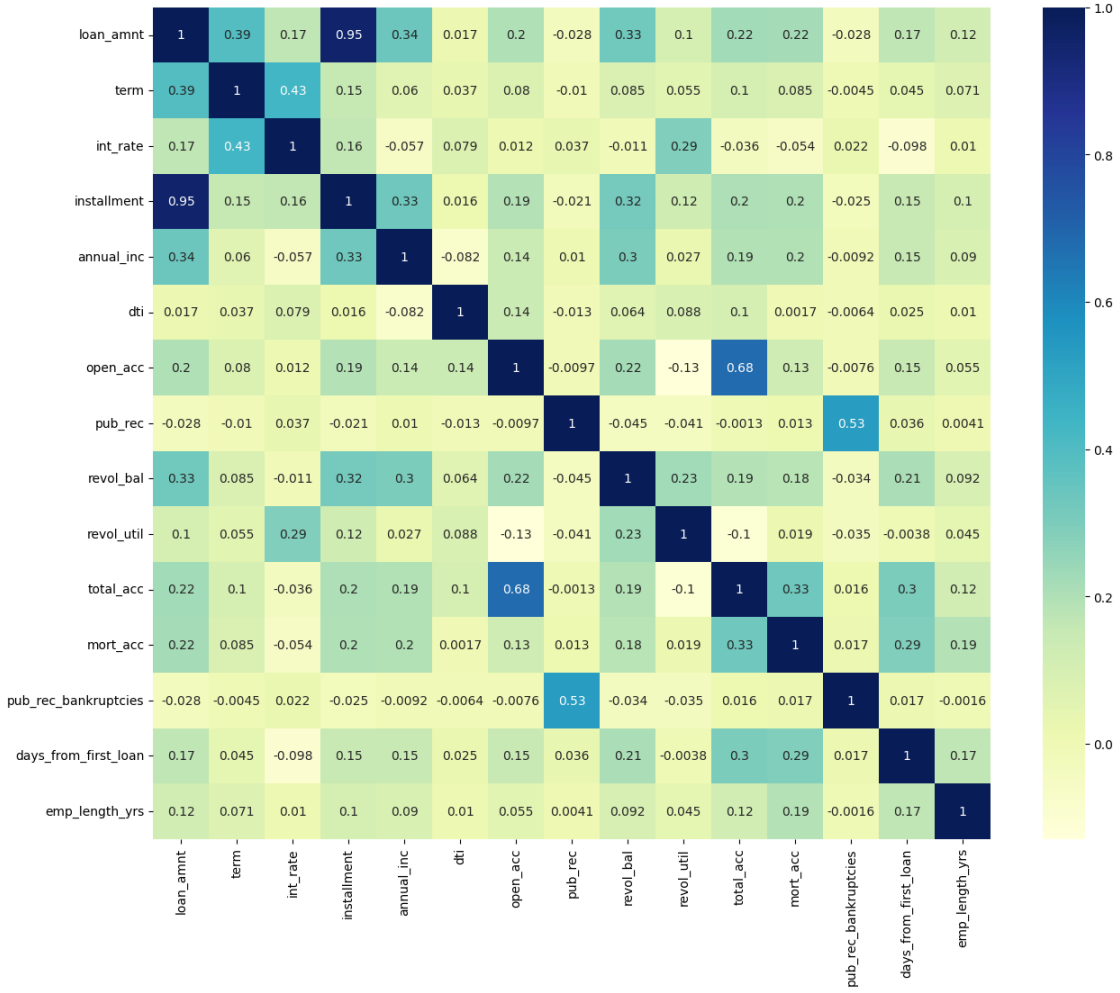
[ ]:
```
sns.pairplot(data = loan_data)
```

[ ]: <seaborn.axisgrid.PairGrid at 0x274291329b0>



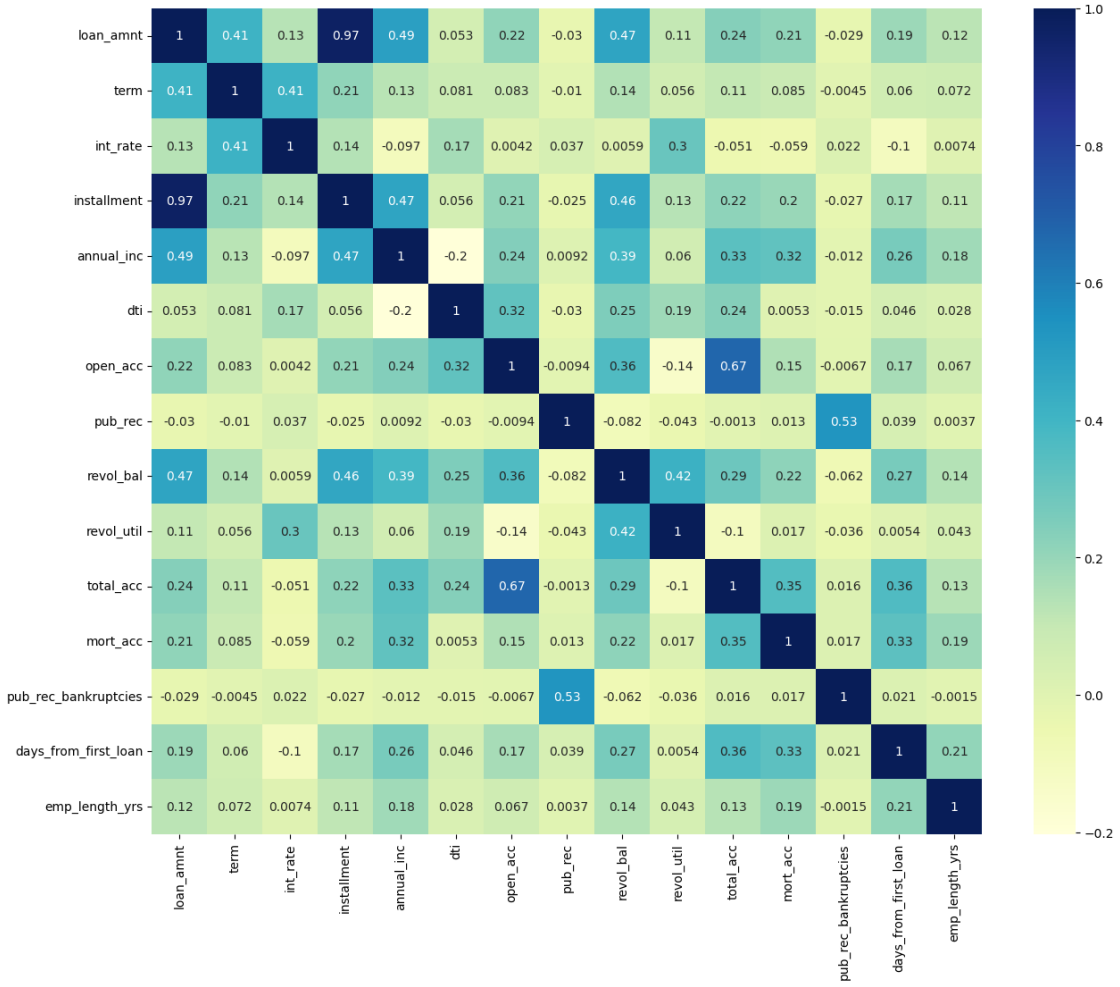
[ ]:
```
fig = plt.figure(figsize= (15,12))
sns.heatmap(loan_data.corr(numeric_only=True), cmap="YlGnBu", annot=True)
```

[ ]: <Axes: >



[ ]:
```
fig = plt.figure(figsize= (15,12))
sns.heatmap(loan_data.corr(numeric_only=True,method = 'spearman'),␣
↪cmap="YlGnBu", annot=True)
```

[ ]: <Axes: >

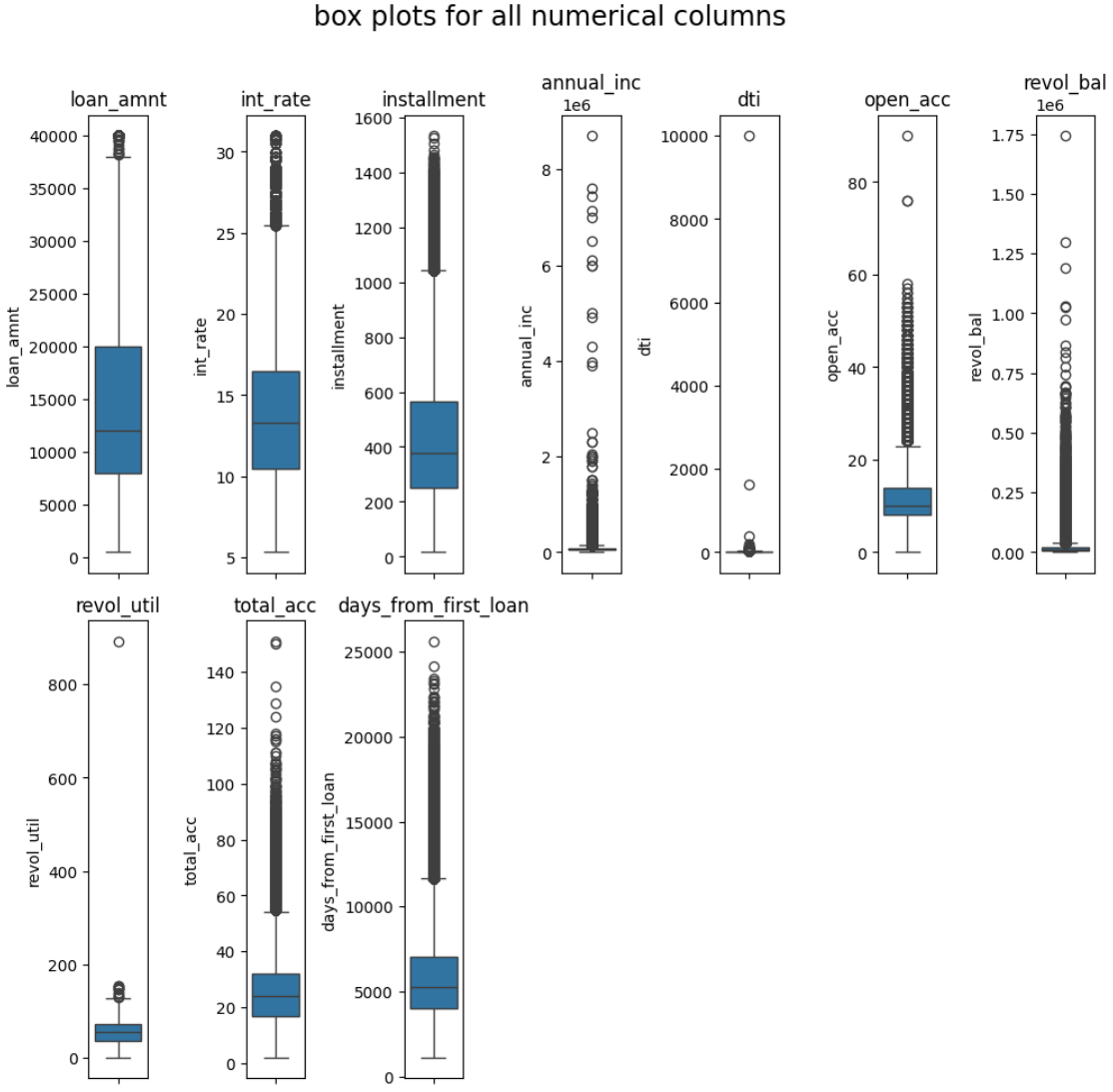


Observations: - Borrowers who defaulted (Charged Off) tend to have higher interest rates (int_rate) and larger loan terms (term) compared to those who fully paid, indicating that longer-term and costlier loans are riskier. - Features like loan_amnt, installment, and dti are slightly higher for Charged Off loans, but the separation is not very sharp, suggesting they contribute moderately to default risk but are not individually strong separators. - Categorical features like grade and sub_grade show clear patterns where lower grades (e.g., E, F, G) have a significantly higher proportion of Charged Off loans compared to higher grades (A, B, C), making credit grade a very important predictor. - The purpose of the loan plays a role: categories like small_business, renewable_energy, and medical have higher Charge Off rates, implying the loan objective itself carries risk information. - Strong multicollinearity is observed between loan_amnt and installment (correlation > 0.95), as well as between open_acc and total_acc, indicating redundant features that need removal or adjustment. - Employment length (emp_length_yrs) and revol_utilization (revol_util) show minimal separation between Fully Paid and Charged Off groups, suggesting these features may be weaker predictors individually unless combined with others.

8 Outlier Treatment

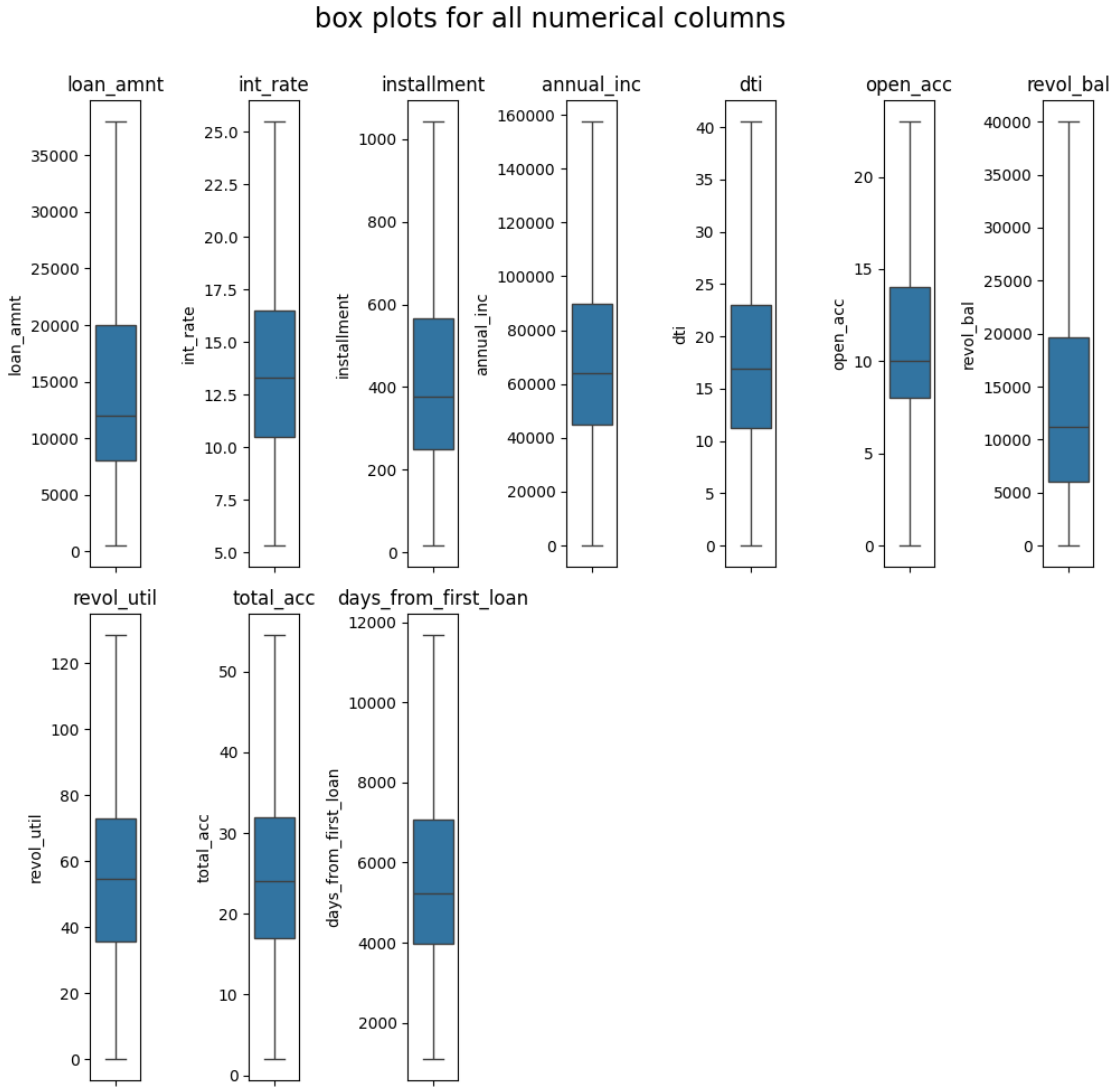
[ ]:
```
numerical_cols = ['loan_amnt', 'int_rate', 'installment', 'annual_inc', 'dti',␣
↪'open_acc', 'revol_bal', 'revol_util', 'total_acc', 'days_from_first_loan']
```

[ ]:
```
fig = plt.figure(figsize = (10,10))
fig.suptitle("box plots for all numerical columns\n",fontsize = "xx-large" )
k = 1
for i in numerical_cols:
    plt.subplot(2,7,k)
    plt.title("{}".format(i))
    sns.boxplot(data=loan_data[i])
    k = k+1
plt.tight_layout()
plt.show()
```



[ ]:
```
def detect_outliers_iqr(df,i):
    q1 = df[i].quantile(0.25)
    q3 = df[i].quantile(0.75)
    IQR = q3-q1
    lwr_bound = q1-(1.5*IQR)
    upr_bound = q3+(1.5*IQR)
    df[i]= pd.DataFrame(np.where(df[i] > upr_bound, upr_bound,(np.where(df[i]␣
 ↪<lwr_bound,lwr_bound, df[i]))), columns=[i])
for i in numerical_cols:
    detect_outliers_iqr(loan_data,i)
```

[ ]:
```
fig = plt.figure(figsize = (10,10))
fig.suptitle("box plots for all numerical columns\n",fontsize = "xx-large" )
k = 1
for i in numerical_cols:
    plt.subplot(2,7,k)
    plt.title("{}".format(i))
    sns.boxplot(data=loan_data[i])
    k = k+1
plt.tight_layout()
plt.show()
```



[ ]:
```
loan_data.drop(['emp_length_missing'], axis=1, inplace=True)
```

[ ]:
```
loan_data.head()
```

[ ]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	\
0	10000.0	36	11.44	329.48	B	B4	
1	8000.0	36	11.99	265.68	B	B5	
2	15600.0	36	10.49	506.97	B	B3	
3	7200.0	36	6.49	220.65	A	A2	
4	24375.0	60	17.27	609.33	C	C5	

	emp_title	home_ownership	annual_inc	verification_status	\
0	Marketing	RENT	117000.0	Not Verified	

```
1          Credit analyst        MORTGAGE     65000.0        Not Verified
2             Statistician            RENT     43057.0     Source Verified
3          Client Advocate            RENT     54000.0        Not Verified
4  Destiny Management Inc.        MORTGAGE     55000.0            Verified

   loan_status             purpose                    title    dti  open_acc  \
0   Fully Paid            vacation                 Vacation  26.24      16.0
1   Fully Paid  debt_consolidation       Debt consolidation  22.05      17.0
2   Fully Paid         credit_card  Credit card refinancing  12.79      13.0
3   Fully Paid         credit_card  Credit card refinancing   2.60       6.0
4  Charged Off         credit_card    Credit Card Refinance  33.95      13.0

   pub_rec  revol_bal  revol_util  total_acc initial_list_status  \
0        0    36369.0        41.8       25.0                   w
1        0    20131.0        53.3       27.0                   f
2        0    11987.0        92.2       26.0                   f
3        0     5472.0        21.5       13.0                   f
4        0    24584.0        69.8       43.0                   f

  application_type  mort_acc  pub_rec_bankruptcies issue_month issue_year  \
0       INDIVIDUAL         0                     0         Jan       2015
1       INDIVIDUAL         1                     0         Jan       2015
2       INDIVIDUAL         0                     0         Jan       2015
3       INDIVIDUAL         0                     0         Nov       2014
4       INDIVIDUAL         0                     0         Apr       2013

  er_cr_line_m er_cr_line_y  days_from_first_loan          state zipcode  \
0          Jun         1990                8980.0       Oklahoma   22690
1          Jul         2004                3836.0   South Dakota   05113
2          Aug         2007                2710.0  West Virginia   05113
3          Sep         2006                2983.0  Massachusetts   00813
4          Mar         1999                5145.0       Virginia   11650

   emp_length_yrs
0              10
1               4
2               1
3               6
4               9
```

[ ]:
```
loan_data['sub_grade'].value_counts()
```

[ ]:
```
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
```

[ ]:
```
loan_data['term'].value_counts()
```

[ ]:
```
term
36    302005
60     94025
Name: count, dtype: int64
```

[ ]:
```
loan_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 396030 entries, 0 to 396029
Data columns (total 31 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
```

```
 0   loan_amnt             396030 non-null  float64
 1   term                  396030 non-null  int64
 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             396030 non-null  object
 7   home_ownership        396030 non-null  object
 8   annual_inc            396030 non-null  float64
 9   verification_status   396030 non-null  object
 10  loan_status           396030 non-null  object
 11  purpose               396030 non-null  object
 12  title                 396030 non-null  object
 13  dti                   396030 non-null  float64
 14  open_acc              396030 non-null  float64
 15  pub_rec               396030 non-null  int64
 16  revol_bal             396030 non-null  float64
 17  revol_util            396030 non-null  float64
 18  total_acc             396030 non-null  float64
 19  initial_list_status   396030 non-null  object
 20  application_type      396030 non-null  object
 21  mort_acc              396030 non-null  int64
 22  pub_rec_bankruptcies  396030 non-null  int64
 23  issue_month           396030 non-null  object
 24  issue_year            396030 non-null  object
 25  er_cr_line_m          396030 non-null  object
 26  er_cr_line_y          396030 non-null  object
 27  days_from_first_loan  396030 non-null  float64
 28  state                 396030 non-null  object
 29  zipcode               396030 non-null  object
 30  emp_length_yrs        396030 non-null  int64
dtypes: float64(10), int64(5), object(16)
memory usage: 93.7+ MB
```

Observations: - Outliers are removed by using IQR + clipping method. - Z-score method can also be used, but the current method is more robust in case skewed distributions.

9 Encoding

9.1 Manual Encoding

[ ]:
```
for i in loan_data.columns:
    print(i)
    print(loan_data[i].nunique())
    if loan_data[i].nunique() <= 10:
        print(loan_data[i].value_counts())
    print('-'*120)
```

```
loan_amnt
1385
--------------------------------------------------------------------------------
----------------------------------------
term
2
term
36    302005
60     94025
Name: count, dtype: int64
--------------------------------------------------------------------------------
----------------------------------------
int_rate
525
--------------------------------------------------------------------------------
----------------------------------------
installment
53616
--------------------------------------------------------------------------------
----------------------------------------
grade
7
grade
B    116018
C    105987
A     64187
D     63524
E     31488
F     11772
G      3054
Name: count, dtype: int64
--------------------------------------------------------------------------------
----------------------------------------
sub_grade
35
--------------------------------------------------------------------------------
----------------------------------------
emp_title
173106
--------------------------------------------------------------------------------
----------------------------------------
home_ownership
6
home_ownership
MORTGAGE    198348
RENT        159790
OWN          37746
OTHER          112
```

```
NONE            31
ANY              3
Name: count, dtype: int64
--------------------------------------------------------------------------------
----------------------------------------
annual_inc
25919
--------------------------------------------------------------------------------
----------------------------------------
verification_status
3
verification_status
Verified           139563
Source Verified    131385
Not Verified       125082
Name: count, dtype: int64
--------------------------------------------------------------------------------
----------------------------------------
loan_status
2
loan_status
Fully Paid     318357
Charged Off     77673
Name: count, dtype: int64
--------------------------------------------------------------------------------
----------------------------------------
purpose
14
--------------------------------------------------------------------------------
----------------------------------------
title
48816
--------------------------------------------------------------------------------
----------------------------------------
dti
4018
--------------------------------------------------------------------------------
----------------------------------------
open_acc
24
--------------------------------------------------------------------------------
----------------------------------------
pub_rec
2
pub_rec
0    388011
1      8019
Name: count, dtype: int64
```

```
--------------------------------------------------------------------------------
----------------------------------------
revol_bal
38808
--------------------------------------------------------------------------------
----------------------------------------
revol_util
1215
--------------------------------------------------------------------------------
----------------------------------------
total_acc
54
--------------------------------------------------------------------------------
----------------------------------------
initial_list_status
2
initial_list_status
f    238066
w    157964
Name: count, dtype: int64
--------------------------------------------------------------------------------
----------------------------------------
application_type
3
application_type
INDIVIDUAL    395319
JOINT            425
DIRECT_PAY       286
Name: count, dtype: int64
--------------------------------------------------------------------------------
----------------------------------------
mort_acc
2
mort_acc
0    237988
1    158042
Name: count, dtype: int64
--------------------------------------------------------------------------------
----------------------------------------
pub_rec_bankruptcies
2
pub_rec_bankruptcies
0    393705
1      2325
Name: count, dtype: int64
--------------------------------------------------------------------------------
----------------------------------------
issue_month
```

```
12
----------------------------------------------------------------------------------------
----------------------------------------
issue_year
10
issue_year
2014    102860
2013     97662
2015     94264
2012     41202
2016     28088
2011     17435
2010      9258
2009      3826
2008      1240
2007       195
Name: count, dtype: int64
----------------------------------------------------------------------------------------
----------------------------------------
er_cr_line_m
12
----------------------------------------------------------------------------------------
----------------------------------------
er_cr_line_y
65
----------------------------------------------------------------------------------------
----------------------------------------
days_from_first_loan
1482
----------------------------------------------------------------------------------------
----------------------------------------
state
54
----------------------------------------------------------------------------------------
----------------------------------------
zipcode
10
zipcode
70466    56985
30723    56546
22690    56527
48052    55917
00813    45824
29597    45471
05113    45402
11650    11226
93700    11151
86630    10981
```

```
Name: count, dtype: int64
----------------------------------------------------------------------------------------
----------------------------------------
emp_length_yrs
10
emp_length_yrs
10    126041
1      75908
2      35827
3      31665
5      26495
4      23952
6      20841
7      20819
8      19168
9      15314
Name: count, dtype: int64
----------------------------------------------------------------------------------------
----------------------------------------
```

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

[ ]:
```
loan_data['loan_status'].value_counts()
```

[ ]:
```
loan_status
0    318357
1     77673
Name: count, dtype: int64
```

[ ]:
```
loan_data['issue_month'].value_counts()
```

[ ]:
```
issue_month
Oct    42130
Jul    39714
Jan    34682
Nov    34068
Apr    33223
Aug    32816
Mar    31919
May    31895
Jun    30140
Dec    29082
Feb    28742
Sep    27619
Name: count, dtype: int64
```

Observations: - Charged off should be `Class 1` and Fully paid should be `Class 0` strictly for

maintaing the correctness in Confusion Matrix. There Manual encoding is used.

9.2 One hot encoding

[ ]:
```
loan_data.dtypes
```

[ ]:
```
loan_amnt               float64
term                      int64
int_rate                float64
installment             float64
grade                    object
sub_grade                object
emp_title                object
home_ownership           object
annual_inc              float64
verification_status      object
loan_status               int64
purpose                  object
title                    object
dti                     float64
open_acc                float64
pub_rec                   int64
revol_bal               float64
revol_util              float64
total_acc               float64
initial_list_status      object
application_type         object
mort_acc                  int64
pub_rec_bankruptcies      int64
issue_month              object
issue_year               object
er_cr_line_m             object
er_cr_line_y             object
days_from_first_loan    float64
state                    object
zipcode                  object
emp_length_yrs            int64
dtype: object
```

[ ]:
```
loan_data.
 ↪drop(['emp_title','title','sub_grade','issue_year','issue_month','er_cr_line_m','er_cr_line
 ↪axis=1, inplace=True)
```

[ ]:
```
loan_data.head()
```

[ ]:
```
   loan_amnt  term  int_rate  installment grade home_ownership  annual_inc  \
0    10000.0    36     11.44       329.48     B           RENT    117000.0
```

```
1     8000.0    36     11.99       265.68     B       MORTGAGE     65000.0
2    15600.0    36     10.49       506.97     B           RENT     43057.0
3     7200.0    36      6.49       220.65     A           RENT     54000.0
4    24375.0    60     17.27       609.33     C       MORTGAGE     55000.0

  verification_status  loan_status             purpose    dti  open_acc  \
0        Not Verified            0            vacation  26.24      16.0
1        Not Verified            0  debt_consolidation  22.05      17.0
2     Source Verified            0         credit_card  12.79      13.0
3        Not Verified            0         credit_card   2.60       6.0
4            Verified            1         credit_card  33.95      13.0

   pub_rec  revol_bal  revol_util  total_acc initial_list_status  \
0        0    36369.0        41.8       25.0                  w
1        0    20131.0        53.3       27.0                  f
2        0    11987.0        92.2       26.0                  f
3        0     5472.0        21.5       13.0                  f
4        0    24584.0        69.8       43.0                  f

  application_type  mort_acc  pub_rec_bankruptcies  days_from_first_loan  \
0       INDIVIDUAL         0                     0                8980.0
1       INDIVIDUAL         1                     0                3836.0
2       INDIVIDUAL         0                     0                2710.0
3       INDIVIDUAL         0                     0                2983.0
4       INDIVIDUAL         0                     0                5145.0

           state zipcode  emp_length_yrs
0       Oklahoma   22690              10
1   South Dakota   05113               4
2  West Virginia   05113               1
3  Massachusetts   00813               6
4       Virginia   11650               9
```

[ ]:
```
# Select categorical columns you want to encode
categorical_cols = ['grade', 'home_ownership', 'verification_status',␣
 ↪'application_type', 'initial_list_status','zipcode']

# Initialize OneHotEncoder
ohe = OneHotEncoder(drop='first', sparse_output=False)

# Fit and transform
encoded_array = ohe.fit_transform(loan_data[categorical_cols])

# Get the encoded feature names
encoded_cols = ohe.get_feature_names_out(categorical_cols)

# Create a DataFrame from the encoded array
```

```
encoded_df = pd.DataFrame(encoded_array, columns=encoded_cols, index=loan_data.
 ↪index)

# Drop the original categorical columns and concatenate the new encoded columns
loan_data_encoded = pd.concat([loan_data.drop(columns=categorical_cols),␣
 ↪encoded_df], axis=1)

# Now loan_data_encoded is ready
```

[ ]:
```
loan_data_encoded.head()
```

[ ]:
```
   loan_amnt  term  int_rate  installment  annual_inc  loan_status  \
0    10000.0    36     11.44       329.48    117000.0            0
1     8000.0    36     11.99       265.68     65000.0            0
2    15600.0    36     10.49       506.97     43057.0            0
3     7200.0    36      6.49       220.65     54000.0            0
4    24375.0    60     17.27       609.33     55000.0            1

              purpose    dti  open_acc  pub_rec  revol_bal  revol_util  \
0            vacation  26.24      16.0        0    36369.0        41.8
1  debt_consolidation  22.05      17.0        0    20131.0        53.3
2         credit_card  12.79      13.0        0    11987.0        92.2
3         credit_card   2.60       6.0        0     5472.0        21.5
4         credit_card  33.95      13.0        0    24584.0        69.8

   total_acc  mort_acc  pub_rec_bankruptcies  days_from_first_loan  \
0       25.0         0                     0                8980.0
1       27.0         1                     0                3836.0
2       26.0         0                     0                2710.0
3       13.0         0                     0                2983.0
4       43.0         0                     0                5145.0

           state  emp_length_yrs  grade_B  grade_C  grade_D  grade_E  grade_F  \
0       Oklahoma              10      1.0      0.0      0.0      0.0      0.0
1   South Dakota               4      1.0      0.0      0.0      0.0      0.0
2  West Virginia               1      1.0      0.0      0.0      0.0      0.0
3  Massachusetts               6      0.0      0.0      0.0      0.0      0.0
4       Virginia               9      0.0      1.0      0.0      0.0      0.0

   grade_G  home_ownership_MORTGAGE  home_ownership_NONE  \
0      0.0                      0.0                  0.0
1      0.0                      1.0                  0.0
2      0.0                      0.0                  0.0
3      0.0                      0.0                  0.0
4      0.0                      1.0                  0.0

   home_ownership_OTHER  home_ownership_OWN  home_ownership_RENT  \
```

```
0                   0.0                 0.0                 1.0
1                   0.0                 0.0                 0.0
2                   0.0                 0.0                 1.0
3                   0.0                 0.0                 1.0
4                   0.0                 0.0                 0.0

   verification_status_Source Verified  verification_status_Verified  \
0                                  0.0                          0.0
1                                  0.0                          0.0
2                                  1.0                          0.0
3                                  0.0                          0.0
4                                  0.0                          1.0

   application_type_INDIVIDUAL  application_type_JOINT  initial_list_status_w  \
0                          1.0                     0.0                    1.0
1                          1.0                     0.0                    0.0
2                          1.0                     0.0                    0.0
3                          1.0                     0.0                    0.0
4                          1.0                     0.0                    0.0

   zipcode_05113  zipcode_11650  zipcode_22690  zipcode_29597  zipcode_30723  \
0            0.0            0.0            1.0            0.0            0.0
1            1.0            0.0            0.0            0.0            0.0
2            1.0            0.0            0.0            0.0            0.0
3            0.0            0.0            0.0            0.0            0.0
4            0.0            1.0            0.0            0.0            0.0

   zipcode_48052  zipcode_70466  zipcode_86630  zipcode_93700
0            0.0            0.0            0.0            0.0
1            0.0            0.0            0.0            0.0
2            0.0            0.0            0.0            0.0
3            0.0            0.0            0.0            0.0
4            0.0            0.0            0.0            0.0
```

[ ]:
```
loan_data_encoded.dtypes
```

[ ]:
```
loan_amnt                          float64
term                                 int64
int_rate                           float64
installment                        float64
annual_inc                         float64
loan_status                          int64
purpose                             object
dti                                float64
open_acc                           float64
pub_rec                              int64
revol_bal                          float64
```

```
revol_util                              float64
total_acc                               float64
mort_acc                                  int64
pub_rec_bankruptcies                      int64
days_from_first_loan                    float64
state                                    object
emp_length_yrs                            int64
grade_B                                 float64
grade_C                                 float64
grade_D                                 float64
grade_E                                 float64
grade_F                                 float64
grade_G                                 float64
home_ownership_MORTGAGE                 float64
home_ownership_NONE                     float64
home_ownership_OTHER                    float64
home_ownership_OWN                      float64
home_ownership_RENT                     float64
verification_status_Source Verified     float64
verification_status_Verified            float64
application_type_INDIVIDUAL             float64
application_type_JOINT                  float64
initial_list_status_w                   float64
zipcode_05113                           float64
zipcode_11650                           float64
zipcode_22690                           float64
zipcode_29597                           float64
zipcode_30723                           float64
zipcode_48052                           float64
zipcode_70466                           float64
zipcode_86630                           float64
zipcode_93700                           float64
dtype: object
```

Observations: - If the features have unique values <=10, One Hot encoding is used.

9.3 Target Encoding

[ ]:
```
target_enc = ['purpose','state']
for col in target_enc:
    te = TargetEncoder()
    loan_data_encoded[col] = te.fit_transform(loan_data_encoded[col],␣
 ↪loan_data_encoded['loan_status'])
```

[ ]:
```
loan_data_encoded.head()
```

[ ]:

	loan_amnt	term	int_rate	installment	annual_inc	loan_status	purpose
0	10000.0	36	11.44	329.48	117000.0	0	0.189233
1	8000.0	36	11.99	265.68	65000.0	0	0.207414
2	15600.0	36	10.49	506.97	43057.0	0	0.167118
3	7200.0	36	6.49	220.65	54000.0	0	0.167118
4	24375.0	60	17.27	609.33	55000.0	1	0.167118

	dti	open_acc	pub_rec	revol_bal	revol_util	total_acc	mort_acc
0	26.24	16.0	0	36369.0	41.8	25.0	0
1	22.05	17.0	0	20131.0	53.3	27.0	1
2	12.79	13.0	0	11987.0	92.2	26.0	0
3	2.60	6.0	0	5472.0	21.5	13.0	0
4	33.95	13.0	0	24584.0	69.8	43.0	0

	pub_rec_bankruptcies	days_from_first_loan	state	emp_length_yrs
0	0	8980.0	0.192013	10
1	0	3836.0	0.197038	4
2	0	2710.0	0.204061	1
3	0	2983.0	0.196098	6
4	0	5145.0	0.195101	9

	grade_B	grade_C	grade_D	grade_E	grade_F	grade_G
0	1.0	0.0	0.0	0.0	0.0	0.0
1	1.0	0.0	0.0	0.0	0.0	0.0
2	1.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	1.0	0.0	0.0	0.0	0.0

	home_ownership_MORTGAGE	home_ownership_NONE	home_ownership_OTHER
0	0.0	0.0	0.0
1	1.0	0.0	0.0
2	0.0	0.0	0.0
3	0.0	0.0	0.0
4	1.0	0.0	0.0

	home_ownership_OWN	home_ownership_RENT
0	0.0	1.0
1	0.0	0.0
2	0.0	1.0
3	0.0	1.0
4	0.0	0.0

	verification_status_Source Verified	verification_status_Verified
0	0.0	0.0
1	0.0	0.0
2	1.0	0.0
3	0.0	0.0

```
4                                   0.0                                1.0

   application_type_INDIVIDUAL  application_type_JOINT  initial_list_status_w  \
0                          1.0                    0.0                    1.0
1                          1.0                    0.0                    0.0
2                          1.0                    0.0                    0.0
3                          1.0                    0.0                    0.0
4                          1.0                    0.0                    0.0

   zipcode_05113  zipcode_11650  zipcode_22690  zipcode_29597  zipcode_30723  \
0            0.0            0.0            1.0            0.0            0.0
1            1.0            0.0            0.0            0.0            0.0
2            1.0            0.0            0.0            0.0            0.0
3            0.0            0.0            0.0            0.0            0.0
4            0.0            1.0            0.0            0.0            0.0

   zipcode_48052  zipcode_70466  zipcode_86630  zipcode_93700
0            0.0            0.0            0.0            0.0
1            0.0            0.0            0.0            0.0
2            0.0            0.0            0.0            0.0
3            0.0            0.0            0.0            0.0
4            0.0            0.0            0.0            0.0
```

[ ]:
```
loan_data_encoded.dtypes
```

[ ]:
```
loan_amnt                           float64
term                                  int64
int_rate                            float64
installment                         float64
annual_inc                          float64
loan_status                           int64
purpose                             float64
dti                                 float64
open_acc                            float64
pub_rec                               int64
revol_bal                           float64
revol_util                          float64
total_acc                           float64
mort_acc                              int64
pub_rec_bankruptcies                  int64
days_from_first_loan                float64
state                               float64
emp_length_yrs                        int64
grade_B                             float64
grade_C                             float64
grade_D                             float64
grade_E                             float64
```

```
grade_F                                 float64
grade_G                                 float64
home_ownership_MORTGAGE                 float64
home_ownership_NONE                     float64
home_ownership_OTHER                    float64
home_ownership_OWN                      float64
home_ownership_RENT                     float64
verification_status_Source Verified     float64
verification_status_Verified            float64
application_type_INDIVIDUAL             float64
application_type_JOINT                  float64
initial_list_status_w                   float64
zipcode_05113                           float64
zipcode_11650                           float64
zipcode_22690                           float64
zipcode_29597                           float64
zipcode_30723                           float64
zipcode_48052                           float64
zipcode_70466                           float64
zipcode_86630                           float64
zipcode_93700                           float64
dtype: object
```

Observations: - Target encoding is used for features with unique values > 10 and Date realted and features with high unique values(like emp_title,title, issue_month etc) are removed form the data.

10 Train test split

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

[ ]:
```
X_tr_cv, X_test, y_tr_cv, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42, stratify=y)
X_train, X_val, y_train, y_val = train_test_split(X_tr_cv, y_tr_cv, test_size=0.
↪25,random_state=42,stratify=y_tr_cv)
print("train data shape: ",X_train.shape)
print("val data shape: ",X_val.shape)
print("test data shape: ",X_test.shape)
```

```
train data shape:  (237618, 42)
val data shape:  (79206, 42)
test data shape:  (79206, 42)
```

Observations: - Dataset is sufficient enough for train, validation and test split. - Used stratified splitting.

11 Standardization

[ ]:
```
StdSclr = StandardScaler()
StdSclr.fit(X_train) # fit on training data
X_train = StdSclr.transform(X_train)
X_val = StdSclr.transform(X_val)
X_test = StdSclr.transform(X_test)
```

12 Removal of Un-Wanted features using `ols method`

[ ]:
```
X_sm = sm.add_constant(np.array(X_train))  #Statmodels default is without␣
 ↪intercept, to add intercept we need to add constant

sm_model = sm.OLS(y_train, X_sm).fit()

print(sm_model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:            loan_status   R-squared:                       0.460
Model:                            OLS   Adj. R-squared:                  0.459
Method:                 Least Squares   F-statistic:                     4810.
Date:                Sun, 27 Apr 2025   Prob (F-statistic):               0.00
Time:                        01:00:13   Log-Likelihood:                -44583.
No. Observations:              237618   AIC:                         8.925e+04
Df Residuals:                  237575   BIC:                         8.970e+04
Df Model:                          42
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.1961      0.001    327.490      0.000       0.195       0.197
x1            -0.0162      0.005     -3.541      0.000      -0.025      -0.007
x2             0.0263      0.001     17.918      0.000       0.023       0.029
x3            -0.0042      0.002     -1.977      0.048      -0.008   -3.61e-05
x4             0.0235      0.004      5.511      0.000       0.015       0.032
x5            -0.0140      0.001    -17.161      0.000      -0.016      -0.012
x6             0.0049      0.001      7.974      0.000       0.004       0.006
x7             0.0174      0.001     24.086      0.000       0.016       0.019
x8             0.0117      0.001     13.077      0.000       0.010       0.013
x9             0.0028      0.001      3.894      0.000       0.001       0.004
x10           -0.0069      0.001     -8.053      0.000      -0.009      -0.005
x11            0.0081      0.001     10.746      0.000       0.007       0.010
x12           -0.0102      0.001    -11.309      0.000      -0.012      -0.008
x13           -0.0011      0.001     -1.534      0.125      -0.003       0.000
x14        -4.998e-05      0.001     -0.071      0.944      -0.001       0.001
```

```
x15            0.0019      0.001      2.779      0.005       0.001       0.003
x16            0.0028      0.001      4.721      0.000       0.002       0.004
x17           -0.0049      0.001     -7.890      0.000      -0.006      -0.004
x18            0.0093      0.001      7.956      0.000       0.007       0.012
x19            0.0263      0.002     15.994      0.000       0.023       0.030
x20            0.0356      0.002     18.988      0.000       0.032       0.039
x21            0.0349      0.002     19.886      0.000       0.031       0.038
x22            0.0240      0.001     16.907      0.000       0.021       0.027
x23            0.0159      0.001     16.883      0.000       0.014       0.018
x24            0.1026      0.103      0.994      0.320      -0.100       0.305
x25            0.0016      0.002      0.879      0.379      -0.002       0.005
x26            0.0036      0.004      1.012      0.311      -0.003       0.011
x27            0.0636      0.061      1.047      0.295      -0.056       0.183
x28            0.1107      0.101      1.093      0.274      -0.088       0.309
x29            0.0073      0.001      9.986      0.000       0.006       0.009
x30            0.0033      0.001      4.351      0.000       0.002       0.005
x31            0.0004      0.001      0.426      0.670      -0.001       0.002
x32           -0.0030      0.001     -3.125      0.002      -0.005      -0.001
x33            0.0008      0.001      1.241      0.215      -0.000       0.002
x34           -0.0002      0.001     -0.262      0.793      -0.002       0.001
x35            0.1580      0.001    238.045      0.000       0.157       0.159
x36            0.0643      0.001     77.536      0.000       0.063       0.066
x37           -0.0003      0.001     -0.347      0.728      -0.002       0.001
x38            0.0644      0.001     77.566      0.000       0.063       0.066
x39            0.0653      0.001     78.873      0.000       0.064       0.067
x40            0.0651      0.001     78.366      0.000       0.064       0.067
x41            0.1553      0.001    234.561      0.000       0.154       0.157
x42            0.1560      0.001    235.691      0.000       0.155       0.157
==============================================================================
Omnibus:                    78969.670   Durbin-Watson:                   1.993
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           203543.011
Skew:                           1.865   Prob(JB):                         0.00
Kurtosis:                       5.579   Cond. No.                         523.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

[ ]:
```
col_names = list(X.columns)
i = 1
for j in col_names:
    if sm_model.pvalues[i] > 0.05:
        print(f"{j:<20} --> Weight = {sm_model.params[i]:<10.5}, p-value =␣
 ↪{sm_model.pvalues[i]:<10.5}")
    i += 1
```

```
mort_acc             --> Weight = -0.0011301, p-value = 0.12503
```

```
pub_rec_bankruptcies --> Weight = -4.9982e-05, p-value = 0.94368
home_ownership_MORTGAGE --> Weight = 0.1026    , p-value = 0.3203
home_ownership_NONE  --> Weight = 0.0016236 , p-value = 0.3792
home_ownership_OTHER --> Weight = 0.0036121 , p-value = 0.31145
home_ownership_OWN   --> Weight = 0.063605  , p-value = 0.29526
home_ownership_RENT  --> Weight = 0.11072   , p-value = 0.27422
application_type_INDIVIDUAL --> Weight = 0.00040965, p-value = 0.67046
initial_list_status_w --> Weight = 0.00076636, p-value = 0.21474
zipcode_05113        --> Weight = -0.00020821, p-value = 0.79348
zipcode_29597        --> Weight = -0.00027647, p-value = 0.72834

C:\Users\sreem\AppData\Local\Temp\ipykernel_34316\1762540305.py:4:
FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a
future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if sm_model.pvalues[i] > 0.05:
C:\Users\sreem\AppData\Local\Temp\ipykernel_34316\1762540305.py:5:
FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a
future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  print(f"{j:<20} --> Weight = {sm_model.params[i]:<10.5}, p-value =
{sm_model.pvalues[i]:<10.5}")
```

[ ]:
```python
X_working = X_train.copy()  # still NumPy array
col_names = list(col_names) # still list of strings
```

[ ]:
```python
# Ensure you have the feature names list
iteration = 1

while True:
    X_sm = sm.add_constant(X_working)  # add intercept
    sm_model = sm.OLS(y_train, X_sm).fit()

    p_values = sm_model.pvalues[1:]  # skip intercept
    max_p_value = p_values.max()

    if max_p_value > 0.05:
        max_p_index = p_values.argmax()  # get integer index of max p-value
        max_p_feature = col_names[max_p_index]  # corresponding feature name

        print(f"Iteration {iteration}:")
        print(f"Removing feature '{max_p_feature}' with p-value {max_p_value:.
 ↪5f}\n")

        # Remove feature from both X_working and col_names
        X_working = np.delete(X_working, max_p_index, axis=1)
        X_val = np.delete(X_val, max_p_index, axis=1)
        X_test = np.delete(X_test, max_p_index, axis=1)
```

```
        col_names.pop(max_p_index)


        iteration += 1
    else:
        print("  All features have p-value  0.05. Backward elimination␣
 ↪completed.\n")
        break

# Final summary after elimination
X_sm_final = sm.add_constant(X_working)
final_model = sm.OLS(y_train, X_sm_final).fit()
print(final_model.summary())

# Final selected features
final_selected_features = col_names
print("\n  Final Selected Features:")
print(final_selected_features)
```

```
Iteration 1:
Removing feature 'pub_rec_bankruptcies' with p-value 0.94368

Iteration 2:
Removing feature 'zipcode_05113' with p-value 0.79335

Iteration 3:
Removing feature 'zipcode_29597' with p-value 0.80240

Iteration 4:
Removing feature 'application_type_INDIVIDUAL' with p-value 0.67036

Iteration 5:
Removing feature 'home_ownership_NONE' with p-value 0.37922

Iteration 6:
Removing feature 'home_ownership_MORTGAGE' with p-value 0.61772

Iteration 7:
Removing feature 'home_ownership_OTHER' with p-value 0.84677

Iteration 8:
Removing feature 'initial_list_status_w' with p-value 0.21703

Iteration 9:
Removing feature 'mort_acc' with p-value 0.15024

  All features have p-value  0.05. Backward elimination completed.
```

OLS Regression Results

Dep. Variable:	loan_status	R-squared:	0.460
Model:	OLS	Adj. R-squared:	0.459
Method:	Least Squares	F-statistic:	6121.
Date:	Sun, 27 Apr 2025	Prob (F-statistic):	0.00
Time:	01:00:20	Log-Likelihood:	-44585.
No. Observations:	237618	AIC:	8.924e+04
Df Residuals:	237584	BIC:	8.959e+04
Df Model:	33		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.1961	0.001	327.492	0.000	0.195	0.197
x1	-0.0162	0.005	-3.535	0.000	-0.025	-0.007
x2	0.0264	0.001	17.995	0.000	0.023	0.029
x3	-0.0046	0.002	-2.167	0.030	-0.009	-0.000
x4	0.0234	0.004	5.508	0.000	0.015	0.032
x5	-0.0140	0.001	-17.342	0.000	-0.016	-0.012
x6	0.0049	0.001	7.976	0.000	0.004	0.006
x7	0.0175	0.001	24.248	0.000	0.016	0.019
x8	0.0119	0.001	13.453	0.000	0.010	0.014
x9	0.0027	0.001	4.549	0.000	0.002	0.004
x10	-0.0070	0.001	-8.172	0.000	-0.009	-0.005
x11	0.0081	0.001	10.750	0.000	0.007	0.010
x12	-0.0105	0.001	-11.982	0.000	-0.012	-0.009
x13	0.0018	0.001	2.637	0.008	0.000	0.003
x14	0.0028	0.001	4.724	0.000	0.002	0.004
x15	-0.0050	0.001	-7.978	0.000	-0.006	-0.004
x16	0.0094	0.001	8.044	0.000	0.007	0.012
x17	0.0265	0.002	16.183	0.000	0.023	0.030
x18	0.0358	0.002	19.184	0.000	0.032	0.039
x19	0.0351	0.002	20.095	0.000	0.032	0.039
x20	0.0242	0.001	17.089	0.000	0.021	0.027
x21	0.0160	0.001	17.026	0.000	0.014	0.018
x22	0.0034	0.001	5.387	0.000	0.002	0.005
x23	0.0105	0.001	15.680	0.000	0.009	0.012
x24	0.0073	0.001	10.026	0.000	0.006	0.009
x25	0.0033	0.001	4.297	0.000	0.002	0.005
x26	-0.0033	0.001	-5.527	0.000	-0.004	-0.002
x27	0.1581	0.001	255.448	0.000	0.157	0.159
x28	0.0645	0.001	97.701	0.000	0.063	0.066
x29	0.0646	0.001	97.853	0.000	0.063	0.066
x30	0.0654	0.001	99.235	0.000	0.064	0.067
x31	0.0653	0.001	98.892	0.000	0.064	0.067
x32	0.1554	0.001	251.210	0.000	0.154	0.157

```
x33            0.1561      0.001    252.520      0.000       0.155       0.157
==============================================================================
Omnibus:                    78973.323   Durbin-Watson:                   1.993
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           203559.952
Skew:                           1.865   Prob(JB):                         0.00
Kurtosis:                       5.579   Cond. No.                         20.2
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.

 Final Selected Features:
['loan_amnt', 'term', 'int_rate', 'installment', 'annual_inc', 'purpose', 'dti',
'open_acc', 'pub_rec', 'revol_bal', 'revol_util', 'total_acc',
'days_from_first_loan', 'state', 'emp_length_yrs', 'grade_B', 'grade_C',
'grade_D', 'grade_E', 'grade_F', 'grade_G', 'home_ownership_OWN',
'home_ownership_RENT', 'verification_status_Source Verified',
'verification_status_Verified', 'application_type_JOINT', 'zipcode_11650',
'zipcode_22690', 'zipcode_30723', 'zipcode_48052', 'zipcode_70466',
'zipcode_86630', 'zipcode_93700']
```

Observations: - P-value > 0.05 is indicating that the even if the weights of those features are made to 0, accuracy wouldnot be effected. - Therefore these are to treated as redundant features, hereby removing them in Train, Validation, Test data.

13 Automatic multicollinearity removal based on VIF

[ ]:
```
vif = pd.DataFrame()
vif['Features'] = col_names
vif['VIF'] = [variance_inflation_factor(X_working, i) for i in range(X_working.
 ↪shape[1])]

vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

[ ]:

	Features	VIF
0	loan_amnt	58.40
3	installment	50.54
2	int_rate	12.30
17	grade_D	9.72
18	grade_E	8.51
16	grade_C	7.47
1	term	5.99
19	grade_F	5.60
15	grade_B	3.77

```
20                                grade_G   2.46
7                               open_acc   2.19
11                             total_acc   2.15
9                              revol_bal   2.03
4                             annual_inc   1.83
24         verification_status_Verified   1.61
10                             revol_util   1.58
23  verification_status_Source Verified   1.48
6                                    dti   1.45
22                   home_ownership_RENT   1.25
12                  days_from_first_loan   1.24
28                         zipcode_30723   1.22
30                         zipcode_70466   1.22
29                         zipcode_48052   1.21
27                         zipcode_22690   1.21
21                     home_ownership_OWN   1.11
14                         emp_length_yrs   1.08
31                         zipcode_86630   1.07
32                         zipcode_93700   1.07
26                         zipcode_11650   1.07
5                                purpose   1.04
8                                pub_rec   1.02
13                                 state   1.00
25                application_type_JOINT   1.00
```

[ ]:
```python
iteration = 1

while True:
    # Calculate VIFs
    vif = pd.DataFrame()
    vif['Features'] = col_names
    vif['VIF'] = [variance_inflation_factor(X_working, i) for i in␣
 ↪range(X_working.shape[1])]

    # Find feature with maximum VIF
    max_vif = vif['VIF'].max()
    max_vif_feature = vif.loc[vif['VIF'].idxmax(), 'Features']

    if max_vif > 10:
        print(f"VIF Iteration {iteration}:")
        print(f"Removing feature '{max_vif_feature}' with VIF = {max_vif:.
 ↪2f}\n")

        # Find column index corresponding to feature
        max_vif_index = col_names.index(max_vif_feature)

        # Remove feature from both X_working and col_names
```

```
        X_working = np.delete(X_working, max_vif_index, axis=1)
        X_val = np.delete(X_val, max_vif_index, axis=1)
        X_test = np.delete(X_test, max_vif_index, axis=1)
        # Remove feature from col_names
        col_names.pop(max_vif_index)

        iteration += 1
    else:
        print("  All features have VIF   10. Multicollinearity removal completed.
 ↪\n")
        break

# Final Output
final_selected_features = col_names
print("  Final Selected Features after p-value and VIF elimination:")
print(final_selected_features)
```

```
VIF Iteration 1:
Removing feature 'loan_amnt' with VIF = 58.40

VIF Iteration 2:
Removing feature 'int_rate' with VIF = 11.73

  All features have VIF   10. Multicollinearity removal completed.

  Final Selected Features after p-value and VIF elimination:
['term', 'installment', 'annual_inc', 'purpose', 'dti', 'open_acc', 'pub_rec',
'revol_bal', 'revol_util', 'total_acc', 'days_from_first_loan', 'state',
'emp_length_yrs', 'grade_B', 'grade_C', 'grade_D', 'grade_E', 'grade_F',
'grade_G', 'home_ownership_OWN', 'home_ownership_RENT',
'verification_status_Source Verified', 'verification_status_Verified',
'application_type_JOINT', 'zipcode_11650', 'zipcode_22690', 'zipcode_30723',
'zipcode_48052', 'zipcode_70466', 'zipcode_86630', 'zipcode_93700']
```

[ ]:
```
X_train = X_working.copy()
```

[ ]:
```
column_names = ['term', 'installment', 'annual_inc', 'purpose', 'dti',␣
 ↪'open_acc', 'pub_rec', 'revol_bal', 'revol_util', 'total_acc',␣
 ↪'days_from_first_loan', 'state', 'emp_length_yrs', 'grade_B', 'grade_C',␣
 ↪'grade_D', 'grade_E', 'grade_F', 'grade_G', 'home_ownership_OWN',␣
 ↪'home_ownership_RENT', 'verification_status_Source Verified',␣
 ↪'verification_status_Verified', 'application_type_JOINT', 'zipcode_11650',␣
 ↪'zipcode_22690', 'zipcode_30723', 'zipcode_48052', 'zipcode_70466',␣
 ↪'zipcode_86630', 'zipcode_93700']
```

[ ]:
```
len(column_names)
```

[ ]: 31

Observations: - Applied the Mutli-collinearity check and removed the features with VIF value >=10.

14 Balancing the data - Smote

[ ]:
```
print("Train data shape: ",X_train.shape, y_train.shape)
```

```
Train data shape:  (237618, 31) (237618,)
```

[ ]:
```
SmoteBL = SMOTE(k_neighbors=5, random_state=42)
X_smote, y_smote = SmoteBL.fit_resample(X_train, y_train)
print("Shape of X_smote: ",X_smote.shape)
print("Shape of y_smote: ",y_smote.shape)
```

```
Shape of X_smote:  (382028, 31)
Shape of y_smote:  (382028,)
```

15 MLFlow Logging Function

[ ]:
```
def auc_plots(model, X, y):
    try:
        y_pred_prob = model.predict_proba(X)[:, 1]
        fpr, tpr, _ = roc_curve(y, y_pred_prob)
        pr, re, _ = precision_recall_curve(y, y_pred_prob)
        roc_auc = auc(fpr, tpr)
        pr_auc = auc(re, pr)
        return fpr, tpr, pr, re, roc_auc, pr_auc
    except Exception as e:
        print(f"Error in auc_plots: {e}")
        return None, None, None, None, None, None
```

[ ]:
```
def plot_learning_curve(model, X, y, run_name):
    try:
        train_sizes, train_scores, validation_scores = learning_curve(model, X,␣
 ↪y, cv=5, n_jobs=-1)
        train_mean = train_scores.mean(axis=1)
        train_std = train_scores.std(axis=1)
        validation_mean = validation_scores.mean(axis=1)
        validation_std = validation_scores.std(axis=1)

        plt.figure(figsize=(10, 6))
        plt.plot(train_sizes, train_mean, 'o-', color='blue', label='Training␣
 ↪score')
```

```
        plt.plot(train_sizes, validation_mean, 'o-', color='red',␣
↪label='Validation score')
        plt.fill_between(train_sizes, train_mean - train_std, train_mean +␣
↪train_std, alpha=0.1, color='blue')
        plt.fill_between(train_sizes, validation_mean - validation_std,␣
↪validation_mean + validation_std, alpha=0.1, color='red')

        plt.xlabel('Training examples')
        plt.ylabel('Score')
        plt.title('Learning Curve')
        plt.legend(loc='best')
        plt.grid(True)

        path = f"{run_name}_learning_curve.png"
        plt.savefig(path)
        plt.show()
        plt.close()
        return path

    except Exception as e:
        print(f"Error in plot_learning_curve: {e}")
        return None
```

[ ]:
```
def mlflow_logging_and_metric_printing(model, run_name, bal_type,
                                       X_train, y_train, y_pred_train,
                                       X_val, y_val, y_pred_val,
                                       X_test, y_test, y_pred_test,
                                       hyper_tuning_score=0,␣
↪feature_names=None, **params):

    mlflow.set_experiment("Loan_Tap_Classification")

    with mlflow.start_run(run_name=run_name):
        try:
            # Log parameters
            if params:
                mlflow.log_params(params)
            mlflow.log_param("bal_type", bal_type)

            datasets = {
                'train': (X_train, y_train, y_pred_train),
                'val': (X_val, y_val, y_pred_val),
                'test': (X_test, y_test, y_pred_test)
            }

            for set_name, (X, y_true, y_pred) in datasets.items():
                metrics = {
```

```
                    f"Accuracy_{set_name}": accuracy_score(y_true, y_pred),
                    f"Precision_{set_name}": precision_score(y_true, y_pred),
                    f"Recall_{set_name}": recall_score(y_true, y_pred),
                    f"F1_score_{set_name}": f1_score(y_true, y_pred),
                    f"F2_score_{set_name}": fbeta_score(y_true, y_pred, beta=2)
                }

                # Print and Log metrics
                print(f"\n{set_name.capitalize()} Metrics:")
                for key, value in metrics.items():
                    print(f"  {key}: {value:.4f}")
                    mlflow.log_metric(key, value)

                # AUC plots and metrics
                fpr, tpr, pr, re, roc_auc, pr_auc = auc_plots(model, X, y_true)

                if roc_auc is not None:
                    mlflow.log_metric(f"Roc_auc_{set_name}", roc_auc)
                    print(f"  Roc_auc_{set_name}: {roc_auc:.4f}")
                if pr_auc is not None:
                    mlflow.log_metric(f"Pr_auc_{set_name}", pr_auc)
                    print(f"  Pr_auc_{set_name}: {pr_auc:.4f}")

                # Classification Report
                clf_report = classification_report(y_true, y_pred)
                print(f"\n{set_name.capitalize()} Classification Report:")
                print(clf_report)

                clf_report_dict = classification_report(y_true, y_pred,␣
↪output_dict=True)
                clf_report_df = pd.DataFrame(clf_report_dict).transpose()

                clf_csv = f"{run_name}_{set_name}_classification_report.csv"
                clf_report_df.to_csv(clf_csv)
                mlflow.log_artifact(clf_csv)

            # Log Tuning Metric
            print(f"\nHyperparameter Tuning Best Validation Score:␣
↪{hyper_tuning_score:.4f}")
            mlflow.log_metric("hyperparameter_tuning_best_val_score",␣
↪hyper_tuning_score)

            # Confusion Matrices
            fig, axes = plt.subplots(1, 3, figsize=(18, 5))
            for i, (set_name, (X, y_true, y_pred)) in enumerate(datasets.
↪items()):
```

```
                cm_display = ConfusionMatrixDisplay(confusion_matrix(y_true,␣
↪y_pred), display_labels=["Class 0", "Class 1"])
                cm_display.plot(ax=axes[i], cmap="Blues")
                axes[i].set_title(f"{set_name.capitalize()} Confusion Matrix")
            plt.tight_layout()
            cm_path = f"{run_name}_confusion_matrices.png"
            plt.savefig(cm_path)
            plt.show()
            plt.close()
            mlflow.log_artifact(cm_path)

            # ROC and PR curves
            fig, axes = plt.subplots(2, 3, figsize=(18, 12))
            for idx, (set_name, (X, y_true, y_pred)) in enumerate(datasets.
↪items()):
                fpr, tpr, pr, re, roc_auc, pr_auc = auc_plots(model, X, y_true)
                if fpr is None:
                    continue

                # ROC curve
                axes[0, idx].plot(fpr, tpr, label=f'AUC={roc_auc:.2f}')
                axes[0, idx].plot([0,1],[0,1],'k--')
                axes[0, idx].set_title(f"{set_name.capitalize()} ROC Curve")
                axes[0, idx].set_xlabel('FPR')
                axes[0, idx].set_ylabel('TPR')
                axes[0, idx].legend()

                # PR curve
                axes[1, idx].plot(re, pr, label=f'PR AUC={pr_auc:.2f}')
                axes[1, idx].set_title(f"{set_name.capitalize()}␣
↪Precision-Recall Curve")
                axes[1, idx].set_xlabel('Recall')
                axes[1, idx].set_ylabel('Precision')
                axes[1, idx].legend()
            plt.tight_layout()
            auc_plot_path = f"{run_name}_auc_pr_plots.png"
            plt.savefig(auc_plot_path)
            plt.show()
            plt.close()
            mlflow.log_artifact(auc_plot_path)

            # Learning Curve
            lc_path = plot_learning_curve(model, X_train, y_train, run_name)
            if lc_path:
                mlflow.log_artifact(lc_path)

            # --- New Part: Feature Importance Logging ---
```

```
            if hasattr(model, "coef_") and feature_names is not None:
                coef_df = pd.DataFrame(data=model.coef_, columns=feature_names).
↪T
                coef_df.columns = ["Coefficient"]
                coef_df = coef_df.sort_values("Coefficient", ascending=False)

                # Save feature importance dataframe
                coef_csv_path = f"{run_name}_feature_importance.csv"
                coef_df.to_csv(coef_csv_path)
                mlflow.log_artifact(coef_csv_path)

                # Plot feature importance
                plt.figure(figsize=(12, 8))
                coef_df.plot(kind='bar')
                plt.title("Feature Importance (Coefficients)")
                plt.tight_layout()
                fi_plot_path = f"{run_name}_feature_importance_plot.png"
                plt.savefig(fi_plot_path)
                plt.show()
                plt.close()
                mlflow.log_artifact(fi_plot_path)

                # Log Intercept
                intercept_value = model.intercept_[0]
                mlflow.log_param("model_intercept", intercept_value)
                print(f"\nModel Intercept: {intercept_value:.4f}")

            # Log the model itself
            mlflow.sklearn.log_model(model, artifact_path=f"{run_name}_model")
            print("\n MLFLOW Logging Completed.")

        except Exception as e:
            print(f"Error during MLflow logging: {e}")
```

Obsrevations: - ML flow is an experimental tracking tool, which will be useful for the comprison of Metrics of the Models exprimented. - Above `mlflow_logging_and_metric_printing` function will be used to log hyperparameters, metrics, artifacts and model with **run__names** (useful for diffrentiation between models).

16 Model Building

16.1 Simple Logistic Regression - Imbalanced Data

16.1.1 Before Thresholding

[ ]:
```
model = LogisticRegression()
model.fit(X_train,y_train)
```

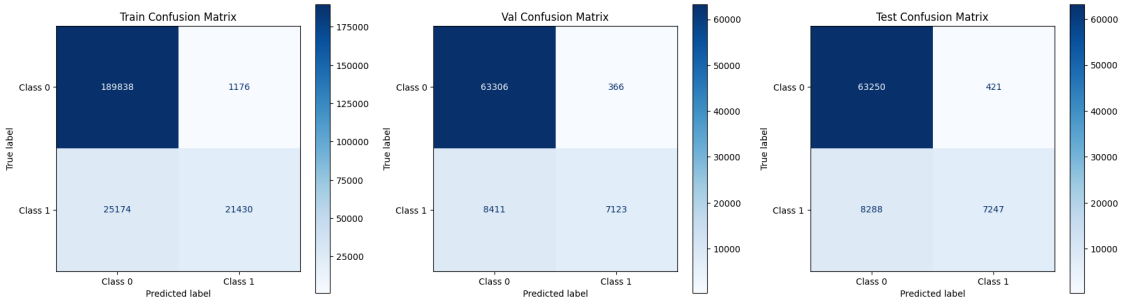
[ ]:
```
LogisticRegression()
```

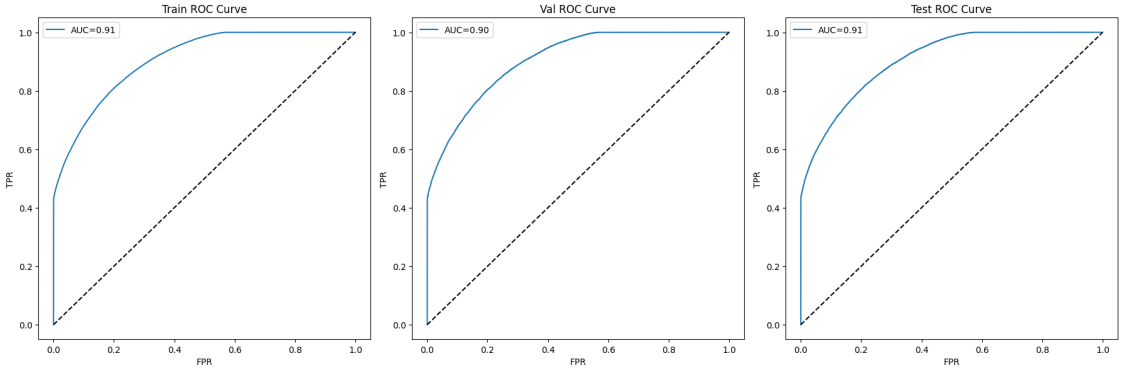
[ ]:
```
y_pred_train = model.predict(X_train)
y_pred_val = model.predict(X_val)
y_pred_test = model.predict(X_test)
```

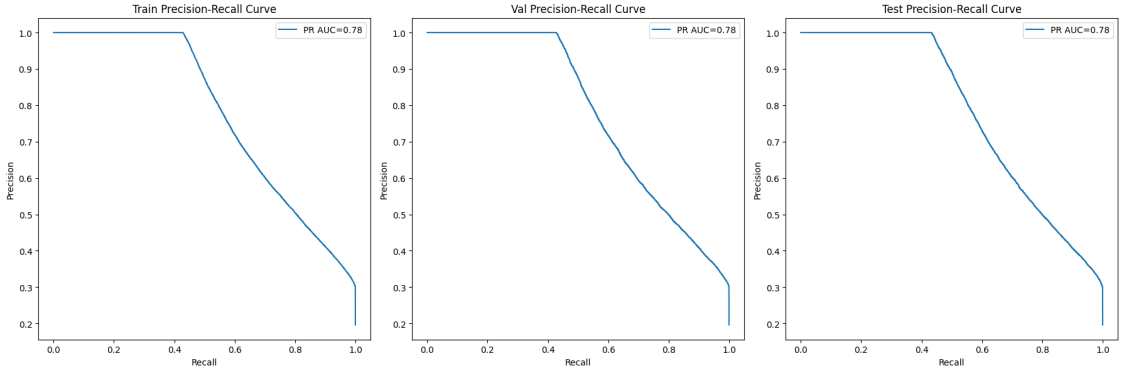
[ ]:
```
feature_names = column_names

mlflow_logging_and_metric_printing(
    model=model,
    run_name="Imbalanced_simple_logistic_regression_before_thresholding",
    bal_type="Imbalanced",
    X_train=X_train, y_train=y_train, y_pred_train=y_pred_train,
    X_val=X_val, y_val=y_val, y_pred_val=y_pred_val,
    X_test=X_test, y_test=y_test, y_pred_test=y_pred_test,
    hyper_tuning_score=0,
    feature_names=feature_names
)
```

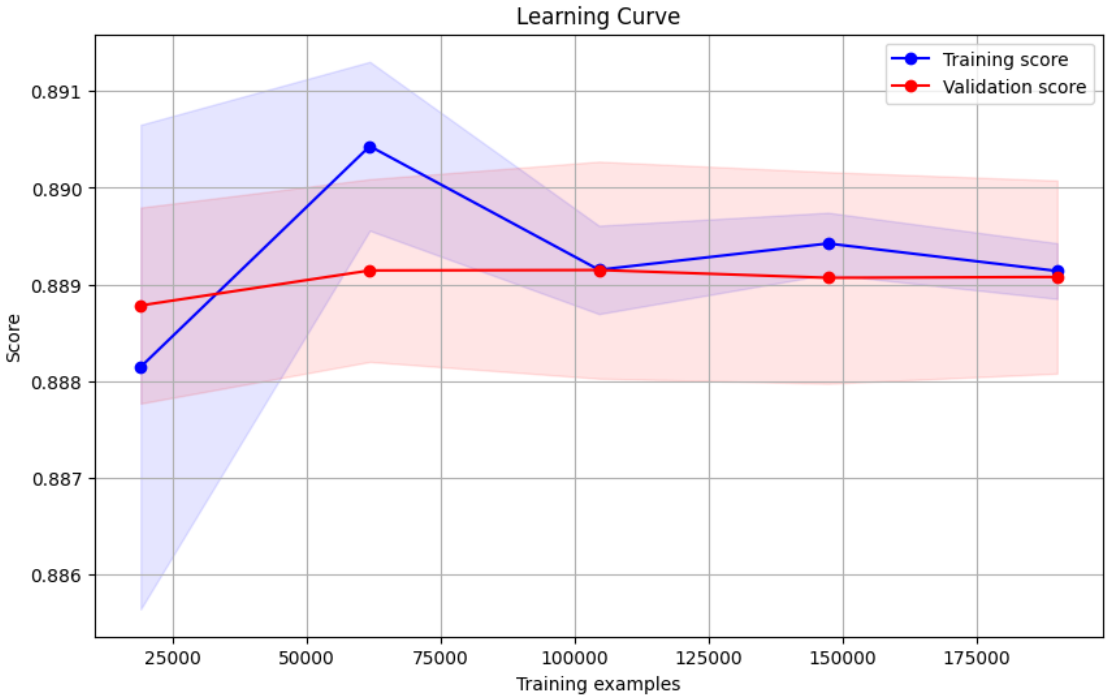
```
Train Metrics:
  Accuracy_train: 0.8891
  Precision_train: 0.9480
  Recall_train: 0.4598
  F1_score_train: 0.6193
  F2_score_train: 0.5126
  Roc_auc_train: 0.9063
  Pr_auc_train: 0.7782

Train Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.99      0.94    191014
           1       0.95      0.46      0.62     46604

    accuracy                           0.89    237618
   macro avg       0.92      0.73      0.78    237618
weighted avg       0.90      0.89      0.87    237618
```

```
Val Metrics:
  Accuracy_val: 0.8892
  Precision_val: 0.9511
  Recall_val: 0.4585
  F1_score_val: 0.6188
  F2_score_val: 0.5115
  Roc_auc_val: 0.9049
  Pr_auc_val: 0.7760

Val Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.99      0.94     63672
           1       0.95      0.46      0.62     15534

    accuracy                           0.89     79206
   macro avg       0.92      0.73      0.78     79206
weighted avg       0.90      0.89      0.87     79206


Test Metrics:
  Accuracy_test: 0.8900
  Precision_test: 0.9451
  Recall_test: 0.4665
  F1_score_test: 0.6247
  F2_score_test: 0.5191
  Roc_auc_test: 0.9061
  Pr_auc_test: 0.7800

Test Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.99      0.94     63671
           1       0.95      0.47      0.62     15535

    accuracy                           0.89     79206
   macro avg       0.91      0.73      0.78     79206
weighted avg       0.90      0.89      0.87     79206


Hyperparameter Tuning Best Validation Score: 0.0000
```

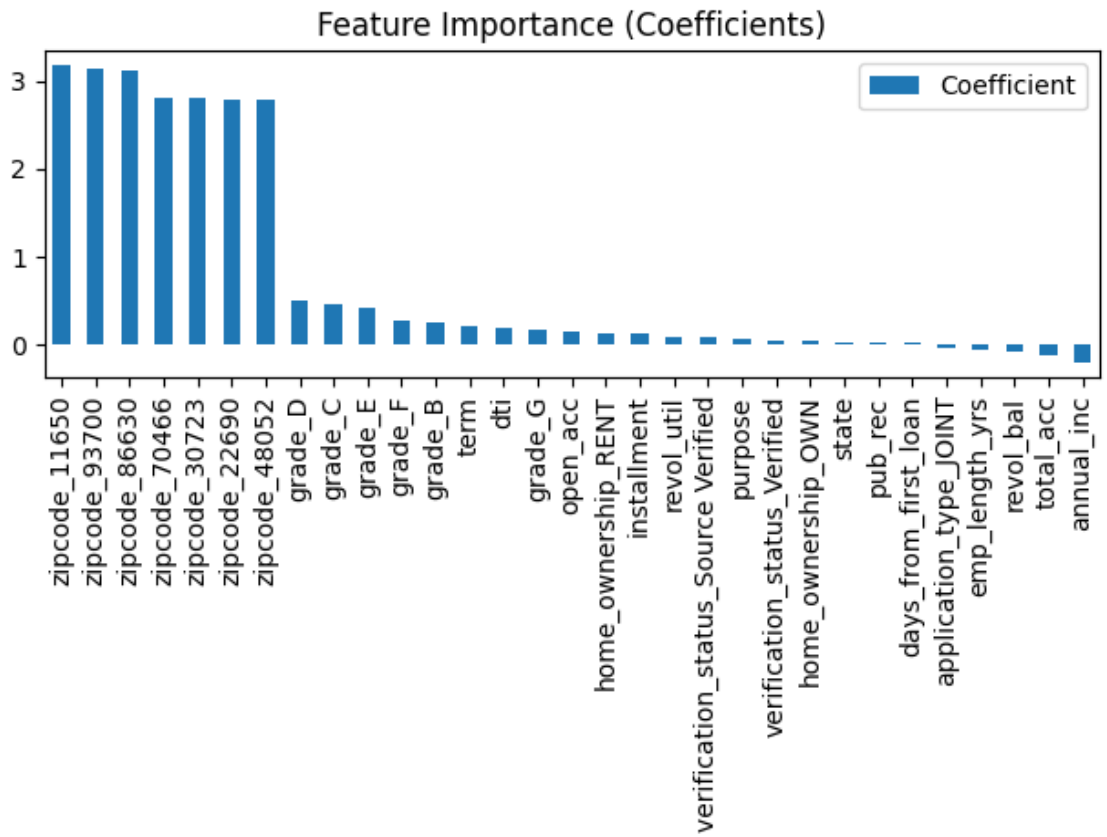

Train Confusion Matrix
Class 0
Class 1
189838
1176
25174
21430
True label
Predicted label
Val Confusion Matrix
63306
366
8411
7123
Test Confusion Matrix
63250
421
8288
7247


Train ROC Curve
AUC=0.91
Val ROC Curve
AUC=0.90
Test ROC Curve
AUC=0.91
TPR
FPR


Train Precision-Recall Curve
PR AUC=0.78
Val Precision-Recall Curve
Test Precision-Recall Curve
Precision
Recall



<Figure size 1200x800 with 0 Axes>




```
Model Intercept: -3.4376

2025/04/27 01:02:02 WARNING mlflow.models.model: Model logged without a
signature and input example. Please set `input_example` parameter when logging
the model to auto infer the model signature.


 MLFLOW Logging Completed.
```

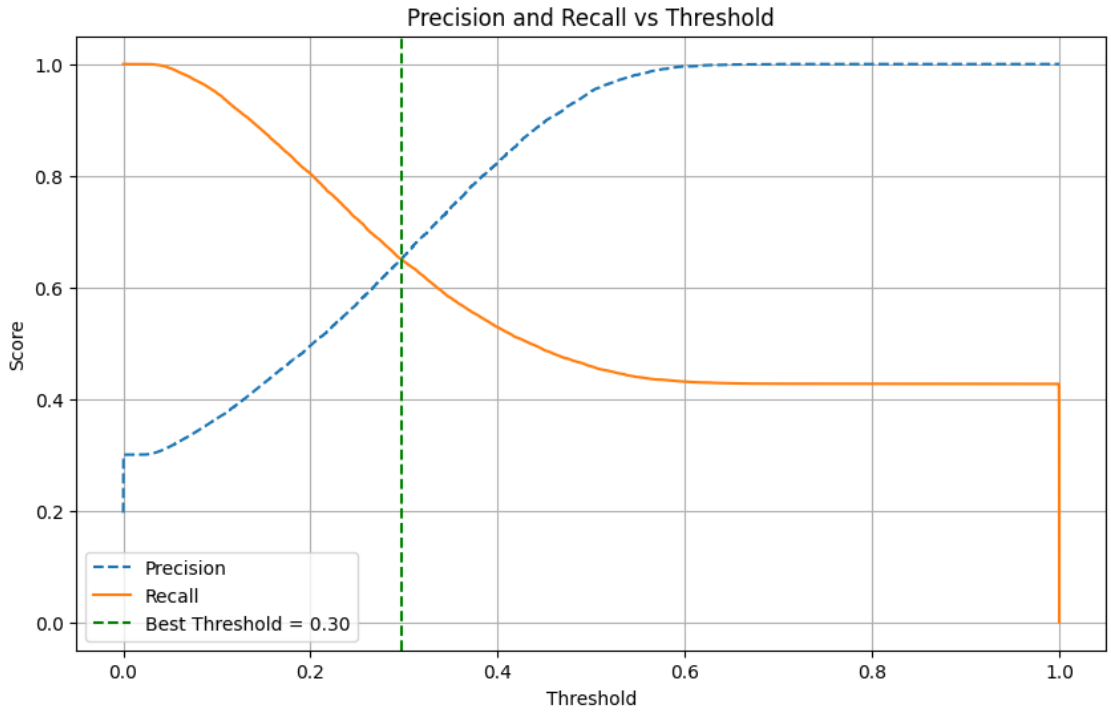
16.1.2 After Thresholding

[ ]:
```
def find_best_threshold(y_true, y_pred_prob):
    precisions, recalls, thresholds = precision_recall_curve(y_true,␣
 ↪y_pred_prob)

    # Find where precision and recall are closest
    differences = np.abs(precisions - recalls)
    best_idx = np.argmin(differences)
    best_threshold = thresholds[best_idx]
```

```
    print(f"Best Threshold where Precision  Recall: {best_threshold:.4f}")

    # Plot Precision-Recall vs Threshold
    threshold_boundary = thresholds.shape[0]  # Because precisions has 1 more␣
↪element than thresholds

    plt.figure(figsize=(10,6))
    plt.plot(thresholds, precisions[:threshold_boundary], label="Precision",␣
↪linestyle="--")
    plt.plot(thresholds, recalls[:threshold_boundary], label="Recall")
    plt.axvline(x=best_threshold, color='green', linestyle='--', label=f'Best␣
↪Threshold = {best_threshold:.2f}')
    plt.xlabel("Threshold")
    plt.ylabel("Score")
    plt.title("Precision and Recall vs Threshold")
    plt.legend(loc='best')
    plt.grid()
    plt.show()

    return best_threshold
```

[ ]:
```
# 2. Predict probabilities
y_pred_proba_train = model.predict_proba(X_train)[:, 1]
y_pred_proba_val = model.predict_proba(X_val)[:, 1]
y_pred_proba_test = model.predict_proba(X_test)[:, 1]
# 3. Find Best Threshold
best_threshold = find_best_threshold(y_val, y_pred_proba_val)
```

```
Best Threshold where Precision  Recall: 0.2970
```



[ ]:
```
# After threshold tuning outside
y_pred_train = (y_pred_proba_train >= best_threshold).astype(int)
y_pred_val = (y_pred_proba_val >= best_threshold).astype(int)
y_pred_test = (y_pred_proba_test >= best_threshold).astype(int)
```

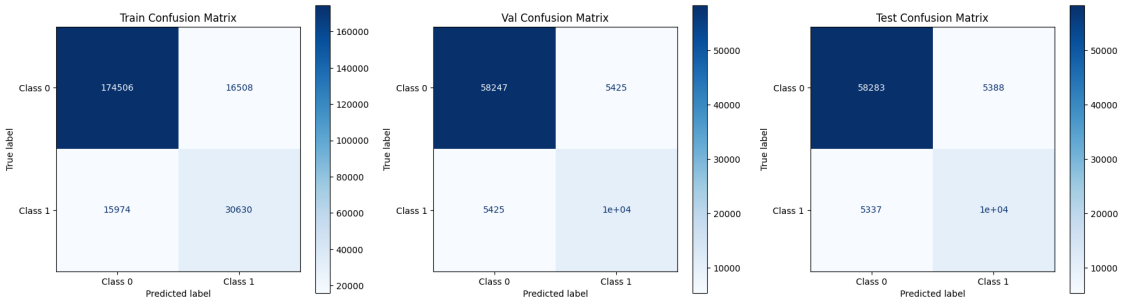
[ ]:
```
mlflow_logging_and_metric_printing(
    model=model,
    run_name="Imbalanced_simple_logistic_regression_after_thresholding",
    bal_type="Imbalanced",
    X_train=X_train, y_train=y_train, y_pred_train=y_pred_train,
    X_val=X_val, y_val=y_val, y_pred_val=y_pred_val,
    X_test=X_test, y_test=y_test, y_pred_test=y_pred_test,
    hyper_tuning_score=0,
    feature_names=feature_names,
    best_threshold=best_threshold
)
```

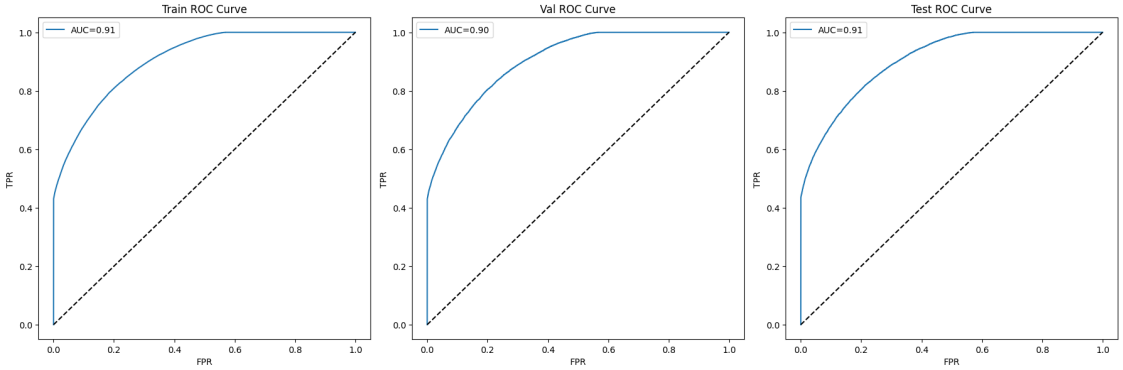
```
Train Metrics:
  Accuracy_train: 0.8633
  Precision_train: 0.6498
  Recall_train: 0.6572
  F1_score_train: 0.6535
  F2_score_train: 0.6557
```

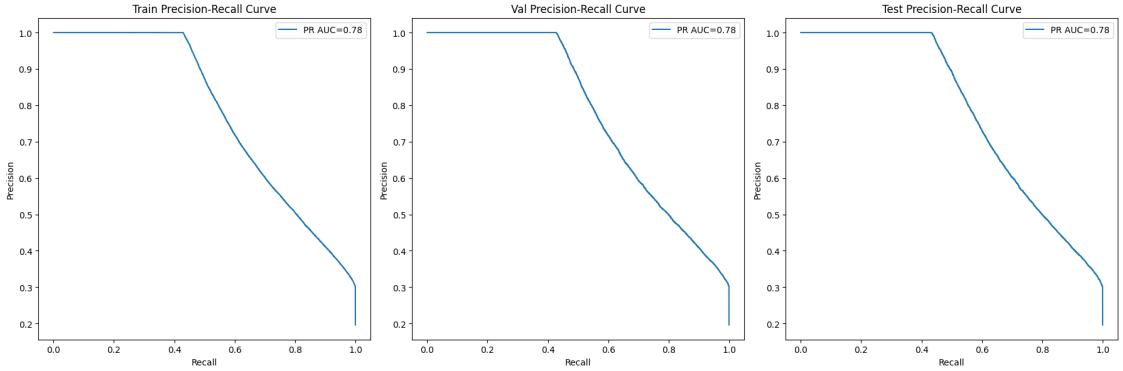
```
  Roc_auc_train: 0.9063
  Pr_auc_train: 0.7782

Train Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.91      0.91    191014
           1       0.65      0.66      0.65     46604

    accuracy                           0.86    237618
   macro avg       0.78      0.79      0.78    237618
weighted avg       0.86      0.86      0.86    237618


Val Metrics:
  Accuracy_val: 0.8630
  Precision_val: 0.6508
  Recall_val: 0.6508
  F1_score_val: 0.6508
  F2_score_val: 0.6508
  Roc_auc_val: 0.9049
  Pr_auc_val: 0.7760

Val Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.91      0.91     63672
           1       0.65      0.65      0.65     15534

    accuracy                           0.86     79206
   macro avg       0.78      0.78      0.78     79206
weighted avg       0.86      0.86      0.86     79206


Test Metrics:
  Accuracy_test: 0.8646
  Precision_test: 0.6543
  Recall_test: 0.6565
  F1_score_test: 0.6554
  F2_score_test: 0.6560
  Roc_auc_test: 0.9061
  Pr_auc_test: 0.7800

Test Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.92      0.92     63671
           1       0.65      0.66      0.66     15535
```

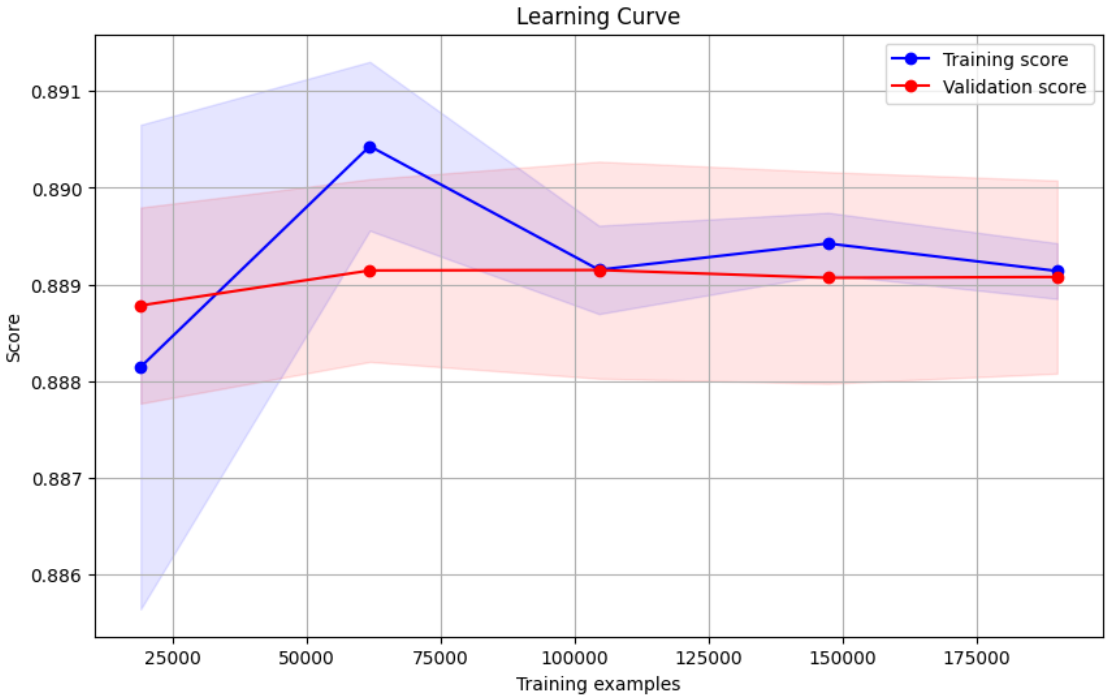
accuracy			0.86	79206
macro avg	0.79	0.79	0.79	79206
weighted avg	0.86	0.86	0.86	79206

Hyperparameter Tuning Best Validation Score: 0.0000

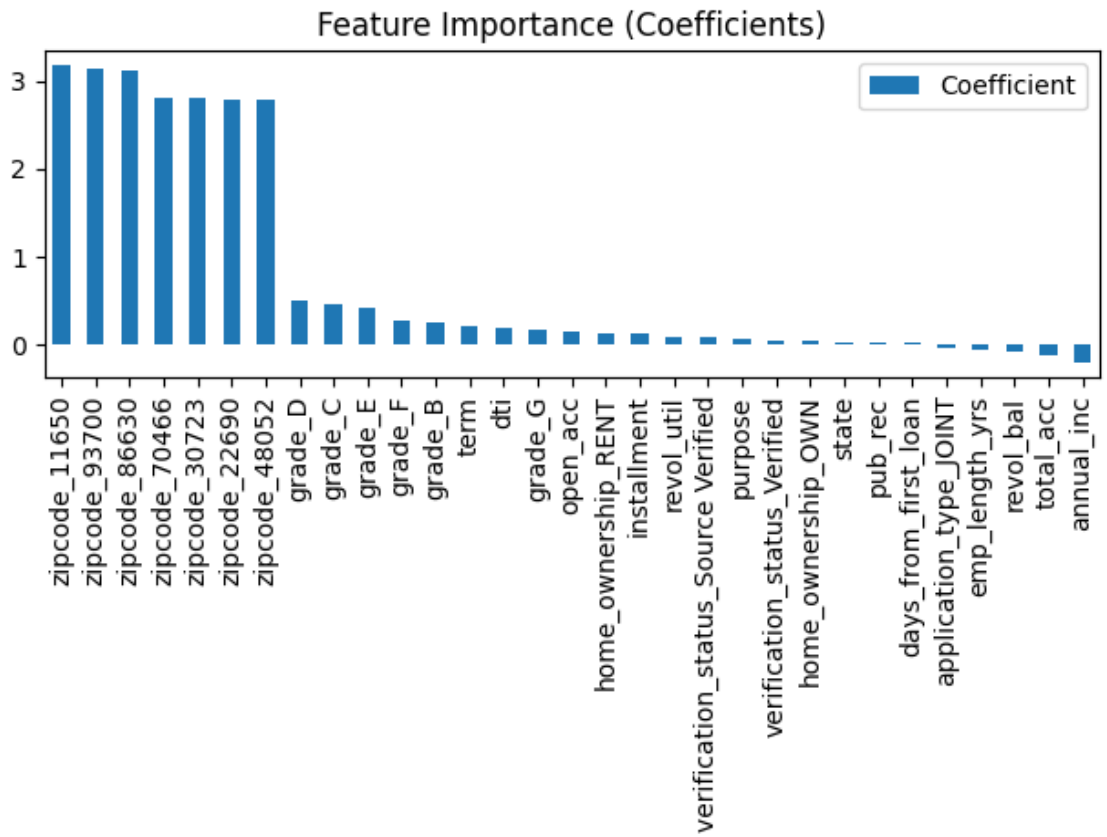








<Figure size 1200x800 with 0 Axes>




```
Model Intercept: -3.4376

2025/04/27 01:02:13 WARNING mlflow.models.model: Model logged without a
signature and input example. Please set `input_example` parameter when logging
the model to auto infer the model signature.


 MLFLOW Logging Completed.
```

16.2 Simple Logistic Regression on Balanced Data

16.2.1 Before Thresholding

[ ]:
```
model = LogisticRegression()
model.fit(X_smote,y_smote)
```

[ ]:
```
LogisticRegression()
```

[ ]:
```
y_pred_smote = model.predict(X_smote)
y_pred_val_smote = model.predict(X_val)
y_pred_test_smote = model.predict(X_test)
```

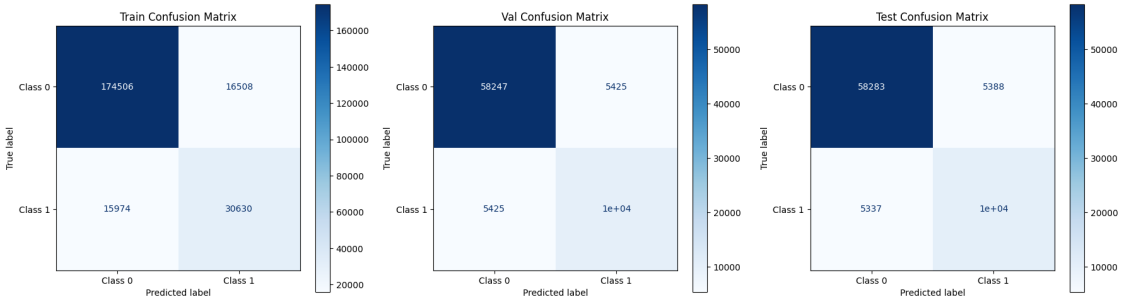
[ ]:
```
feature_names = column_names

mlflow_logging_and_metric_printing(
    model=model,
    run_name="Balanced_simple_logistic_regression_before_thresholding",
    bal_type="Balanced_Smote",
    X_train=X_train, y_train=y_train, y_pred_train=y_pred_train,
    X_val=X_val, y_val=y_val, y_pred_val=y_pred_val,
    X_test=X_test, y_test=y_test, y_pred_test=y_pred_test,
    hyper_tuning_score=0,
    feature_names=feature_names
)
```

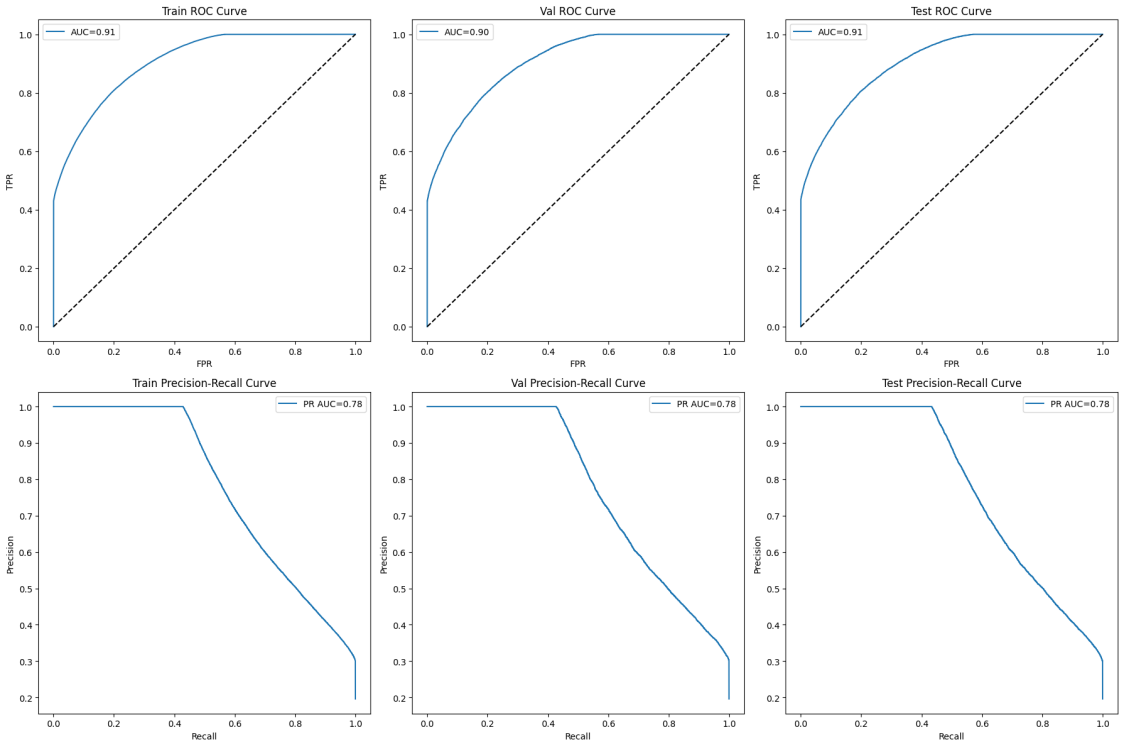
```
Train Metrics:
  Accuracy_train: 0.8633
  Precision_train: 0.6498
  Recall_train: 0.6572
  F1_score_train: 0.6535
  F2_score_train: 0.6557
  Roc_auc_train: 0.9060
  Pr_auc_train: 0.7777

Train Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.91      0.91    191014
           1       0.65      0.66      0.65     46604

    accuracy                           0.86    237618
   macro avg       0.78      0.79      0.78    237618
weighted avg       0.86      0.86      0.86    237618


Val Metrics:
  Accuracy_val: 0.8630
  Precision_val: 0.6508
  Recall_val: 0.6508
  F1_score_val: 0.6508
  F2_score_val: 0.6508
  Roc_auc_val: 0.9046
  Pr_auc_val: 0.7754

Val Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.91      0.91     63672
```

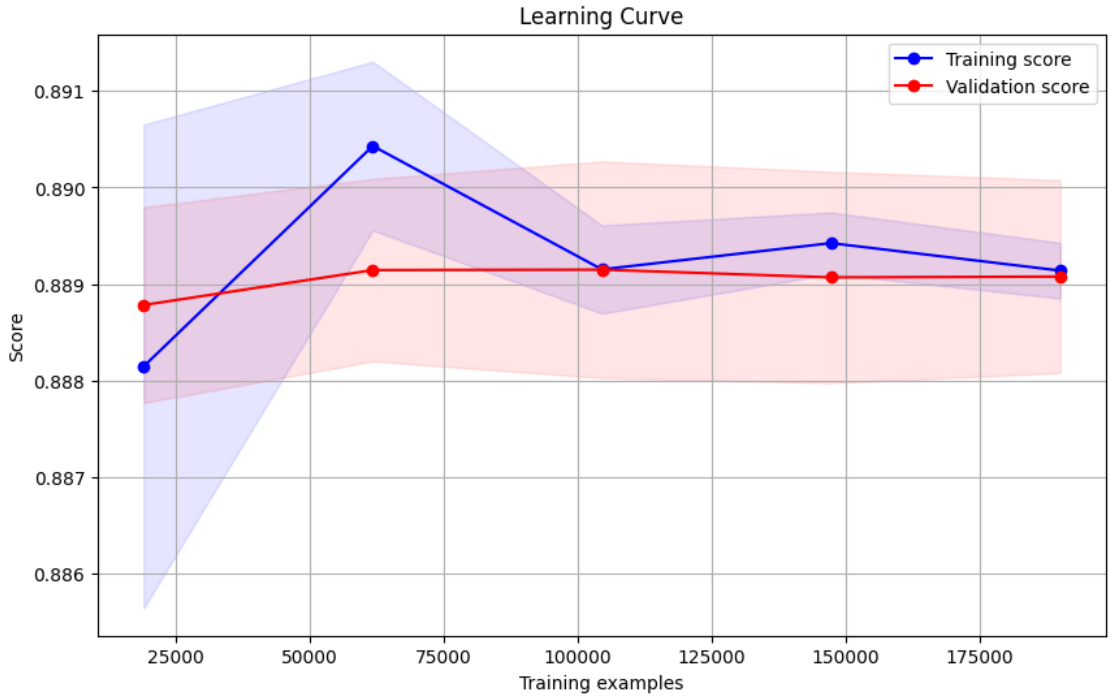
```
           1       0.65      0.65      0.65     15534

    accuracy                           0.86     79206
   macro avg       0.78      0.78      0.78     79206
weighted avg       0.86      0.86      0.86     79206
```

```
Test Metrics:
  Accuracy_test: 0.8646
  Precision_test: 0.6543
  Recall_test: 0.6565
  F1_score_test: 0.6554
  F2_score_test: 0.6560
  Roc_auc_test: 0.9057
  Pr_auc_test: 0.7790

Test Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.92      0.92     63671
           1       0.65      0.66      0.66     15535

    accuracy                           0.86     79206
   macro avg       0.79      0.79      0.79     79206
weighted avg       0.86      0.86      0.86     79206
```

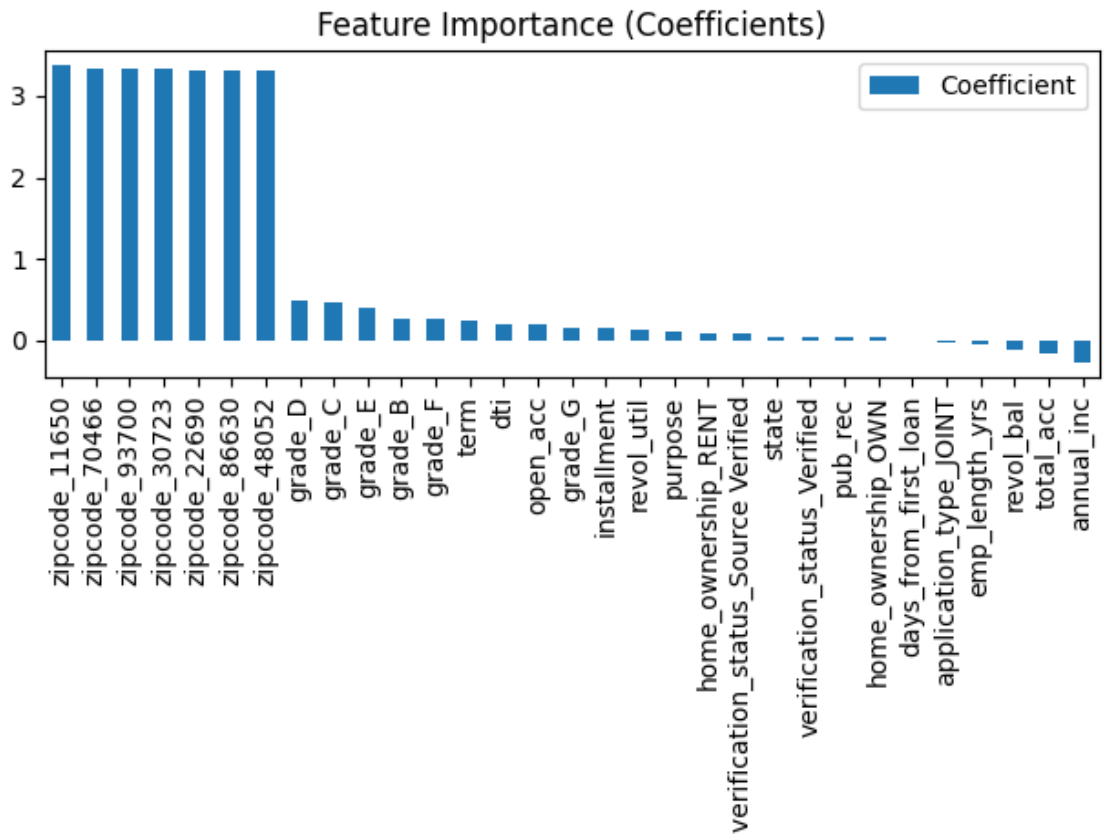
```
Hyperparameter Tuning Best Validation Score: 0.0000
```







<Figure size 1200x800 with 0 Axes>

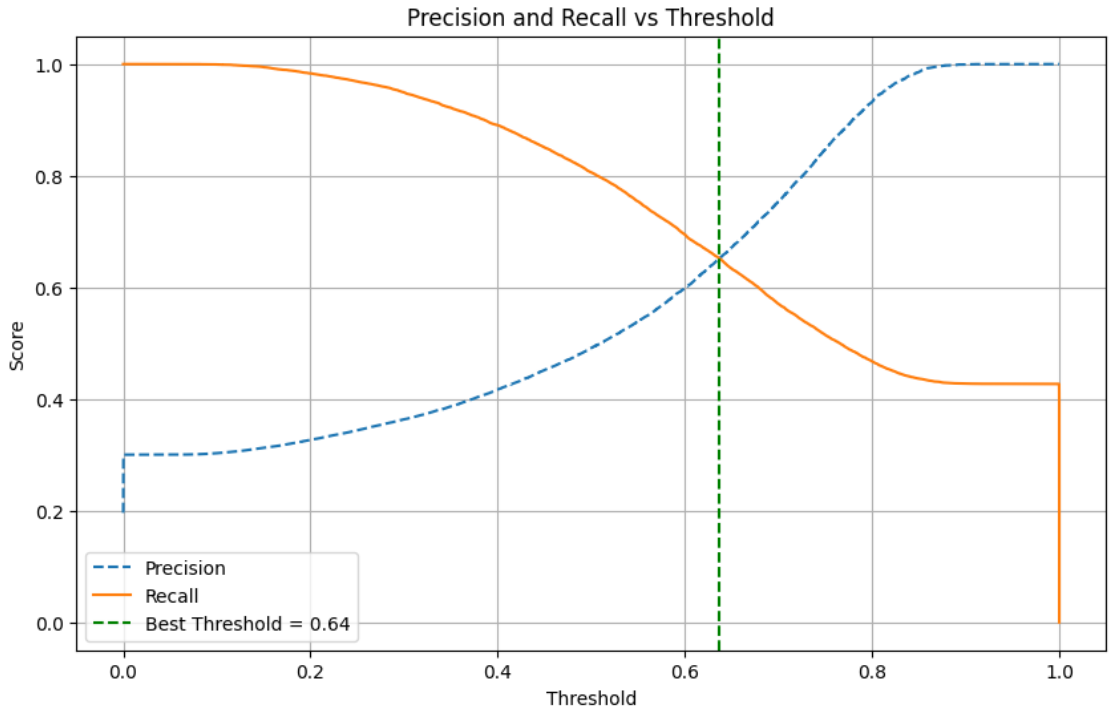



```
Model Intercept: -2.6004

2025/04/27 01:02:24 WARNING mlflow.models.model: Model logged without a
signature and input example. Please set `input_example` parameter when logging
the model to auto infer the model signature.


 MLFLOW Logging Completed.
```

16.2.2 after thresholding

[ ]:
```
# 2. Predict probabilities
y_pred_proba_smote = model.predict_proba(X_smote)[:, 1]
y_pred_proba_val = model.predict_proba(X_val)[:, 1]
y_pred_proba_test = model.predict_proba(X_test)[:, 1]
# 3. Find Best Threshold
best_threshold = find_best_threshold(y_val, y_pred_proba_val)
```

```
Best Threshold where Precision  Recall: 0.6368
```



[ ]:
```
# After threshold tuning outside
y_pred_smote = (y_pred_proba_smote >= best_threshold).astype(int)
y_pred_val = (y_pred_proba_val >= best_threshold).astype(int)
y_pred_test = (y_pred_proba_test >= best_threshold).astype(int)
```

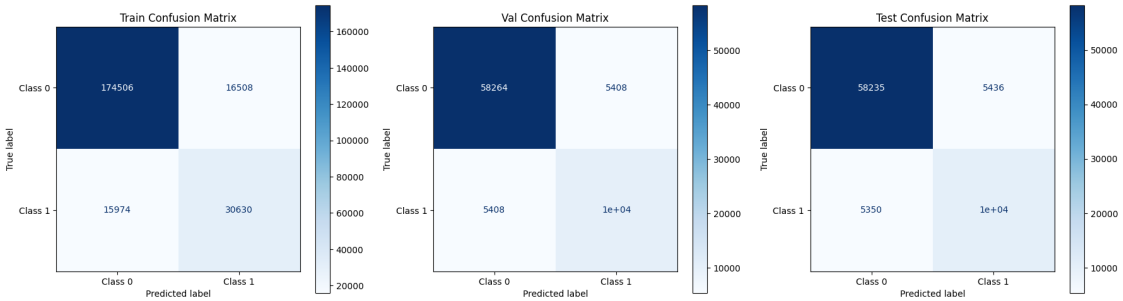
[ ]:
```
feature_names = column_names

mlflow_logging_and_metric_printing(
    model=model,
    run_name="Balanced_simple_logistic_regression_after_thresholding",
    bal_type="Balanced_smote",
    X_train=X_train, y_train=y_train, y_pred_train=y_pred_train,
    X_val=X_val, y_val=y_val, y_pred_val=y_pred_val,
    X_test=X_test, y_test=y_test, y_pred_test=y_pred_test,
    hyper_tuning_score=0,
    feature_names=feature_names,
    best_threshold=best_threshold
)
```

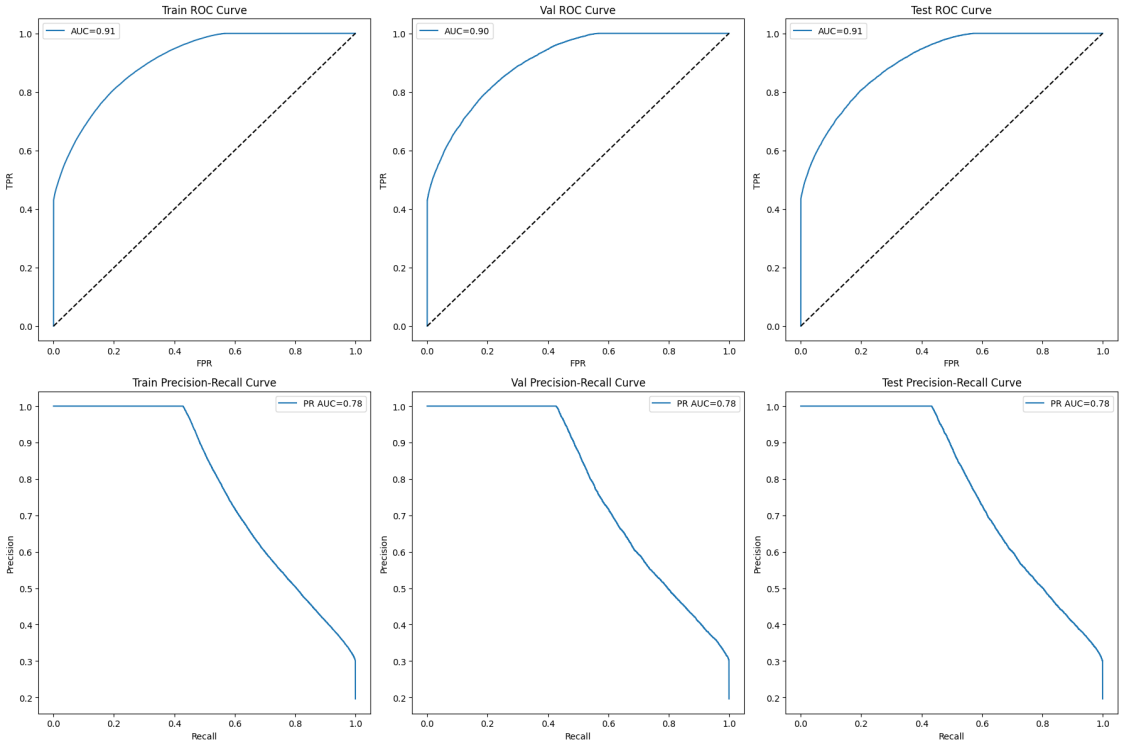
```
Train Metrics:
  Accuracy_train: 0.8633
  Precision_train: 0.6498
  Recall_train: 0.6572
```

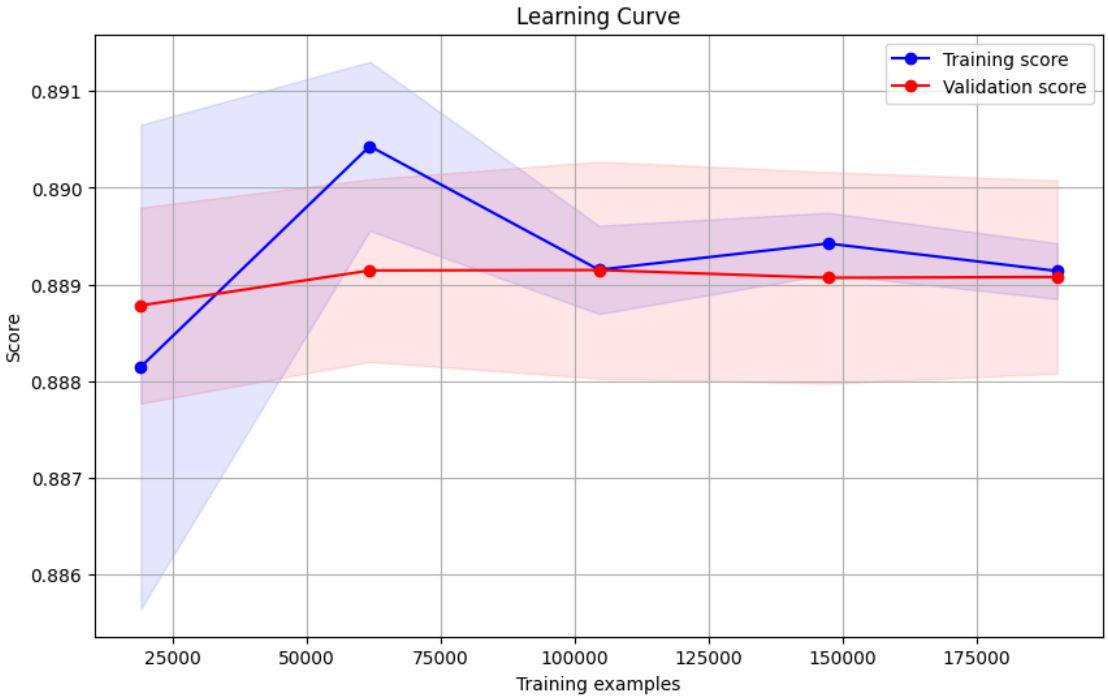
```
  F1_score_train: 0.6535
  F2_score_train: 0.6557
  Roc_auc_train: 0.9060
  Pr_auc_train: 0.7777

Train Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.91      0.91    191014
           1       0.65      0.66      0.65     46604

    accuracy                           0.86    237618
   macro avg       0.78      0.79      0.78    237618
weighted avg       0.86      0.86      0.86    237618


Val Metrics:
  Accuracy_val: 0.8634
  Precision_val: 0.6519
  Recall_val: 0.6519
  F1_score_val: 0.6519
  F2_score_val: 0.6519
  Roc_auc_val: 0.9046
  Pr_auc_val: 0.7754

Val Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.92      0.92     63672
           1       0.65      0.65      0.65     15534

    accuracy                           0.86     79206
   macro avg       0.78      0.78      0.78     79206
weighted avg       0.86      0.86      0.86     79206


Test Metrics:
  Accuracy_test: 0.8638
  Precision_test: 0.6520
  Recall_test: 0.6556
  F1_score_test: 0.6538
  F2_score_test: 0.6549
  Roc_auc_test: 0.9057
  Pr_auc_test: 0.7790

Test Classification Report:
              precision    recall  f1-score   support
```

0	0.92	0.91	0.92	63671
1	0.65	0.66	0.65	15535
accuracy			0.86	79206
macro avg	0.78	0.79	0.78	79206
weighted avg	0.86	0.86	0.86	79206

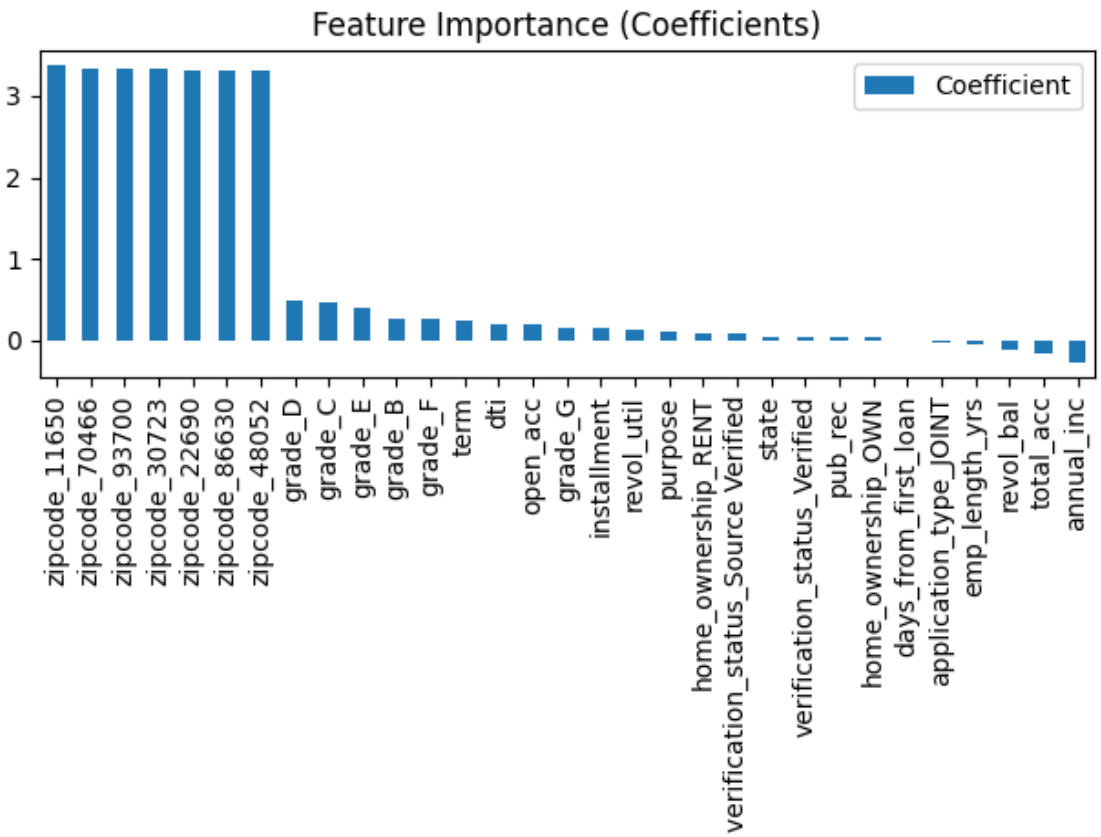
Hyperparameter Tuning Best Validation Score: 0.0000







<Figure size 1200x800 with 0 Axes>




```
Model Intercept: -2.6004

2025/04/27 01:02:34 WARNING mlflow.models.model: Model logged without a
signature and input example. Please set `input_example` parameter when logging
the model to auto infer the model signature.


 MLFLOW Logging Completed.
```

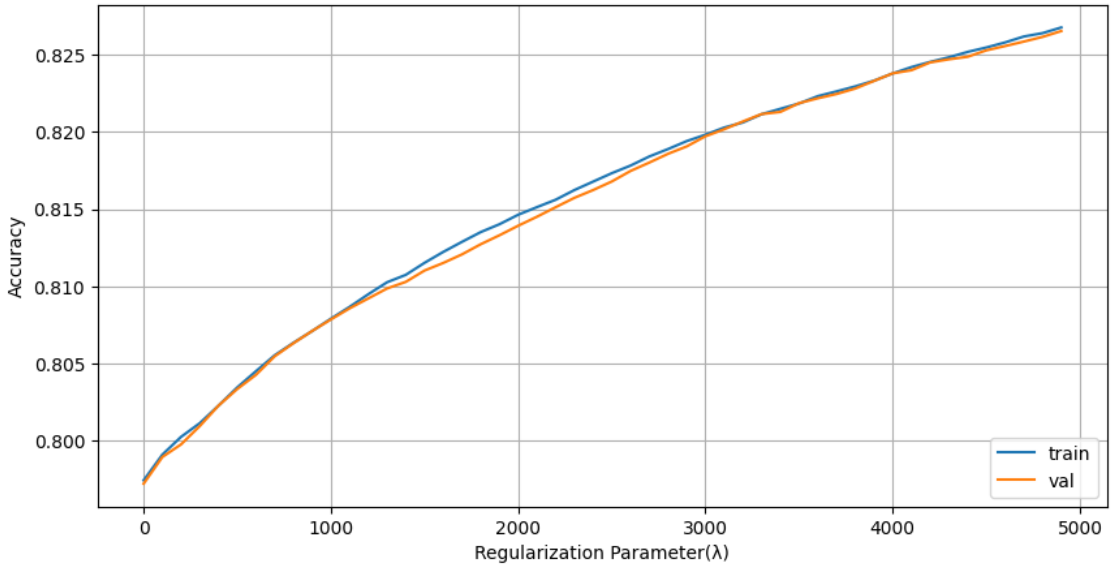
17 Hyperparameter Tuning for Lambda.

17.1 Hyperparamter Tuning on Imbalanced data

[ ]:
```
train_scores = []
val_scores = []
for la in np.arange(0.01, 5000.0, 100): # range of values of Lambda
  model = LogisticRegression(C=1/la,class_weight="balanced")
  model.fit(X_train, y_train)
  train_score = accuracy_score(y_train, model.predict(X_train))
  val_score = accuracy_score(y_val, model.predict(X_val))
```

```
  train_scores.append(train_score)
  val_scores.append(val_score)
```

[ ]:
```
plt.figure(figsize=(10,5))
plt.plot(list(np.arange(0.01, 5000.0, 100)), train_scores, label="train")
plt.plot(list(np.arange(0.01, 5000.0, 100)), val_scores, label="val")
plt.legend(loc='lower right')
plt.xlabel("Regularization Parameter( )")
plt.ylabel("Accuracy")
plt.grid()
plt.show()
```



[ ]:
```
# Find best lambda
lambdas = np.arange(0.01, 5000.0, 100)
best_lambda_index = np.argmax(val_scores)
best_lambda = lambdas[best_lambda_index]
best_val_score = val_scores[best_lambda_index]

print(f"Best Lambda ( ): {best_lambda}")
print(f"Best Validation Accuracy: {best_val_score:.4f}")
```

```
Best Lambda ( ): 4900.01
Best Validation Accuracy: 0.8265
```

17.2 Regularised Logistic Regression - Imbalanced Data

[ ]:
```
model = LogisticRegression(C=1/4900,class_weight="balanced")
model.fit(X_train,y_train)
```

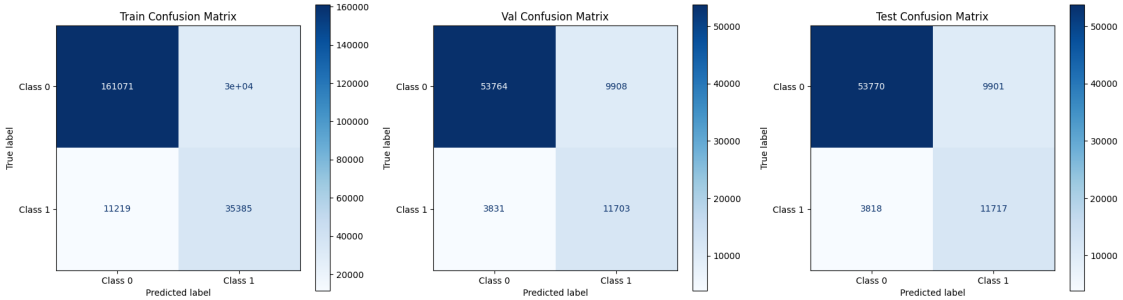
[ ]:
```
LogisticRegression(C=0.00020408163265306123, class_weight='balanced')
```

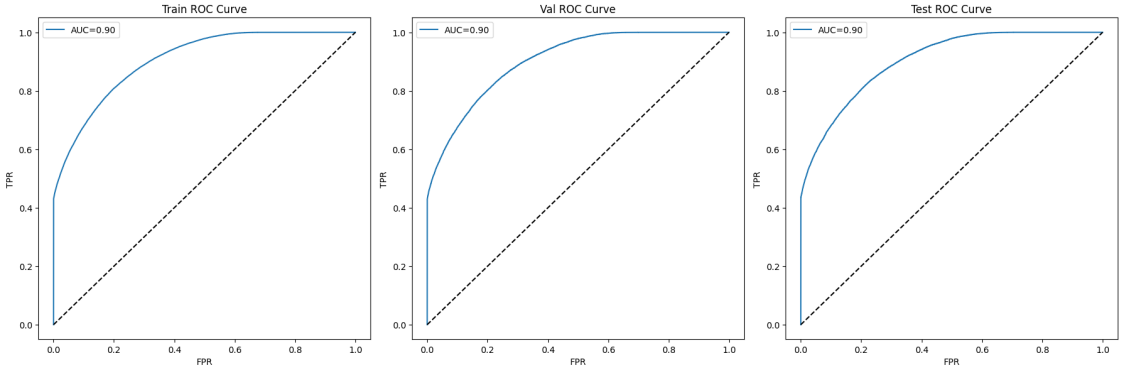
[ ]:
```
y_pred_train = model.predict(X_train)
y_pred_val = model.predict(X_val)
y_pred_test = model.predict(X_test)
```

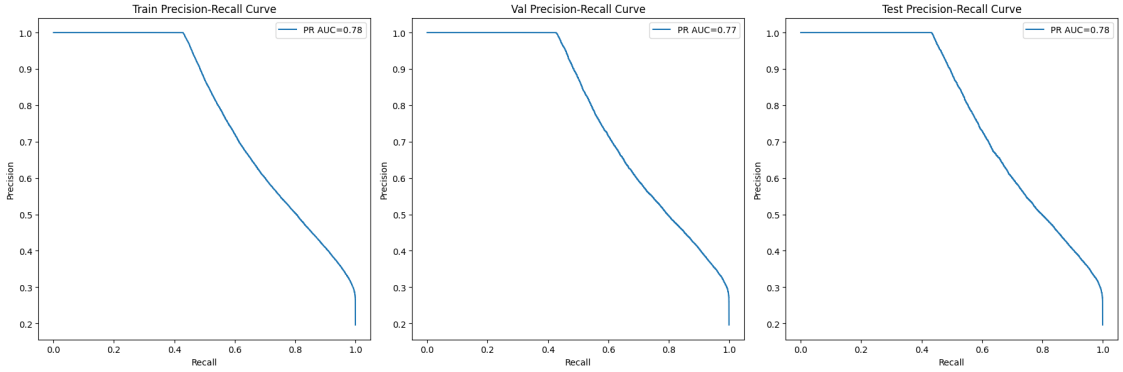
[ ]:
```
feature_names = column_names

mlflow_logging_and_metric_printing(
    model=model,
    run_name="Imbalanced_Regularized_logistic_regression_before_thresholding",
    bal_type="Imbalanced",
    X_train=X_train, y_train=y_train, y_pred_train=y_pred_train,
    X_val=X_val, y_val=y_val, y_pred_val=y_pred_val,
    X_test=X_test, y_test=y_test, y_pred_test=y_pred_test,
    hyper_tuning_score=0,
    feature_names=feature_names,
    best_lambda=best_lambda
)
```

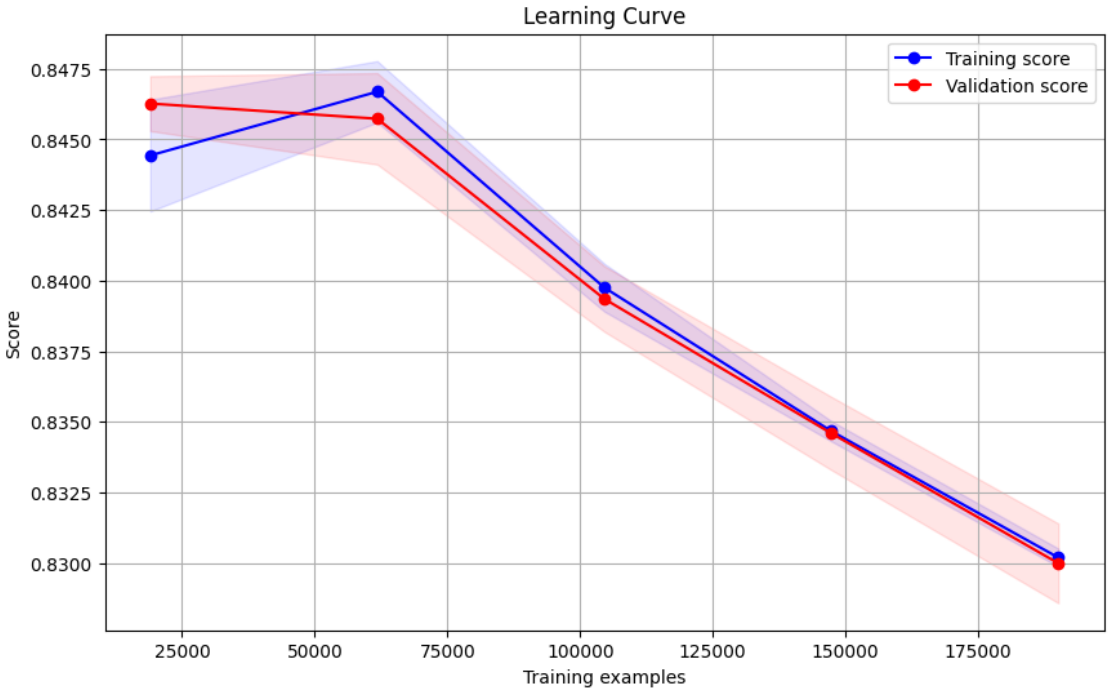
```
Train Metrics:
  Accuracy_train: 0.8268
  Precision_train: 0.5417
  Recall_train: 0.7593
  F1_score_train: 0.6323
  F2_score_train: 0.7028
  Roc_auc_train: 0.9043
  Pr_auc_train: 0.7769

Train Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.84      0.89    191014
           1       0.54      0.76      0.63     46604

    accuracy                           0.83    237618
   macro avg       0.74      0.80      0.76    237618
weighted avg       0.86      0.83      0.84    237618


Val Metrics:
  Accuracy_val: 0.8265
```

```
  Precision_val: 0.5415
  Recall_val: 0.7534
  F1_score_val: 0.6301
  F2_score_val: 0.6987
  Roc_auc_val: 0.9029
  Pr_auc_val: 0.7748

Val Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.84      0.89     63672
           1       0.54      0.75      0.63     15534

    accuracy                           0.83     79206
   macro avg       0.74      0.80      0.76     79206
weighted avg       0.86      0.83      0.84     79206


Test Metrics:
  Accuracy_test: 0.8268
  Precision_test: 0.5420
  Recall_test: 0.7542
  F1_score_test: 0.6307
  F2_score_test: 0.6995
  Roc_auc_test: 0.9040
  Pr_auc_test: 0.7786

Test Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.84      0.89     63671
           1       0.54      0.75      0.63     15535

    accuracy                           0.83     79206
   macro avg       0.74      0.80      0.76     79206
weighted avg       0.86      0.83      0.84     79206


Hyperparameter Tuning Best Validation Score: 0.0000
```

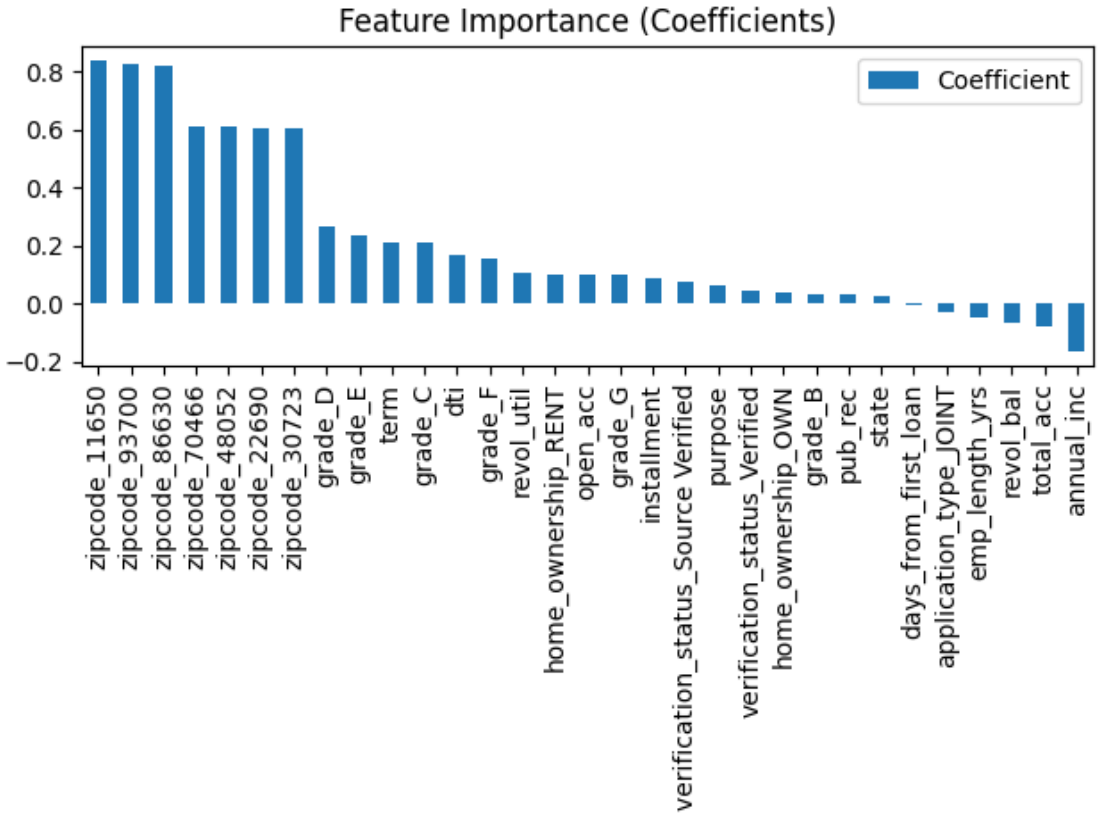

Train Confusion Matrix
Class 0
Class 1
161071
3e+04
11219
35385
True label
Predicted label
Val Confusion Matrix
53764
9908
3831
11703
Test Confusion Matrix
53770
9901
3818
11717


Train ROC Curve
AUC=0.90
TPR
FPR
Val ROC Curve
Test ROC Curve


Train Precision-Recall Curve
PR AUC=0.78
Precision
Recall
Val Precision-Recall Curve
PR AUC=0.77
Test Precision-Recall Curve



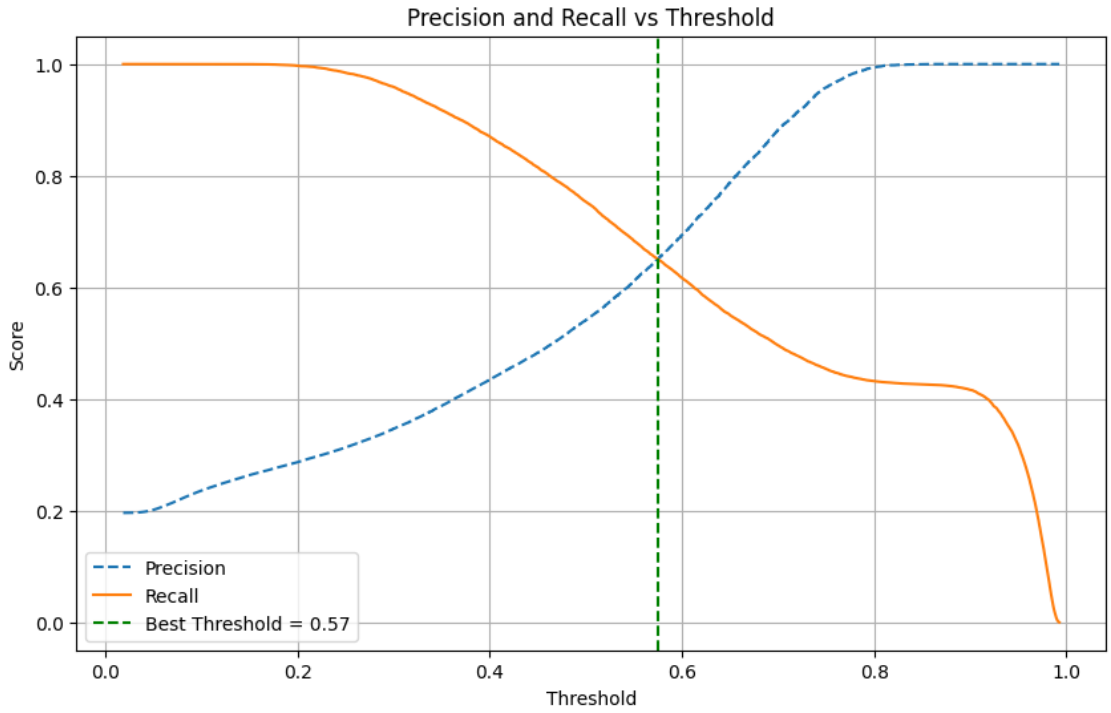
<Figure size 1200x800 with 0 Axes>



```
Model Intercept: -0.6082

2025/04/27 01:02:57 WARNING mlflow.models.model: Model logged without a
signature and input example. Please set `input_example` parameter when logging
the model to auto infer the model signature.


 MLFLOW Logging Completed.
```

[ ]:
```python
# 2. Predict probabilities
y_pred_proba_train = model.predict_proba(X_train)[:, 1]
y_pred_proba_val = model.predict_proba(X_val)[:, 1]
y_pred_proba_test = model.predict_proba(X_test)[:, 1]
# 3. Find Best Threshold
best_threshold = find_best_threshold(y_val, y_pred_proba_val)
```

```
Best Threshold where Precision  Recall: 0.5746
```



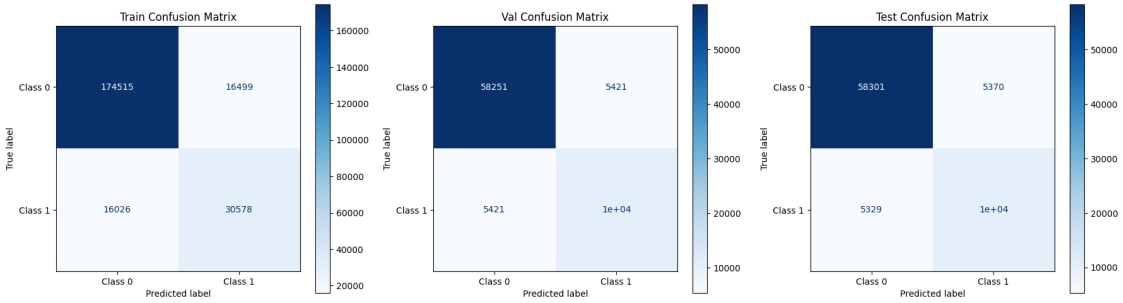
[ ]:
```python
# After threshold tuning outside
y_pred_train = (y_pred_proba_train >= best_threshold).astype(int)
y_pred_val = (y_pred_proba_val >= best_threshold).astype(int)
y_pred_test = (y_pred_proba_test >= best_threshold).astype(int)
```

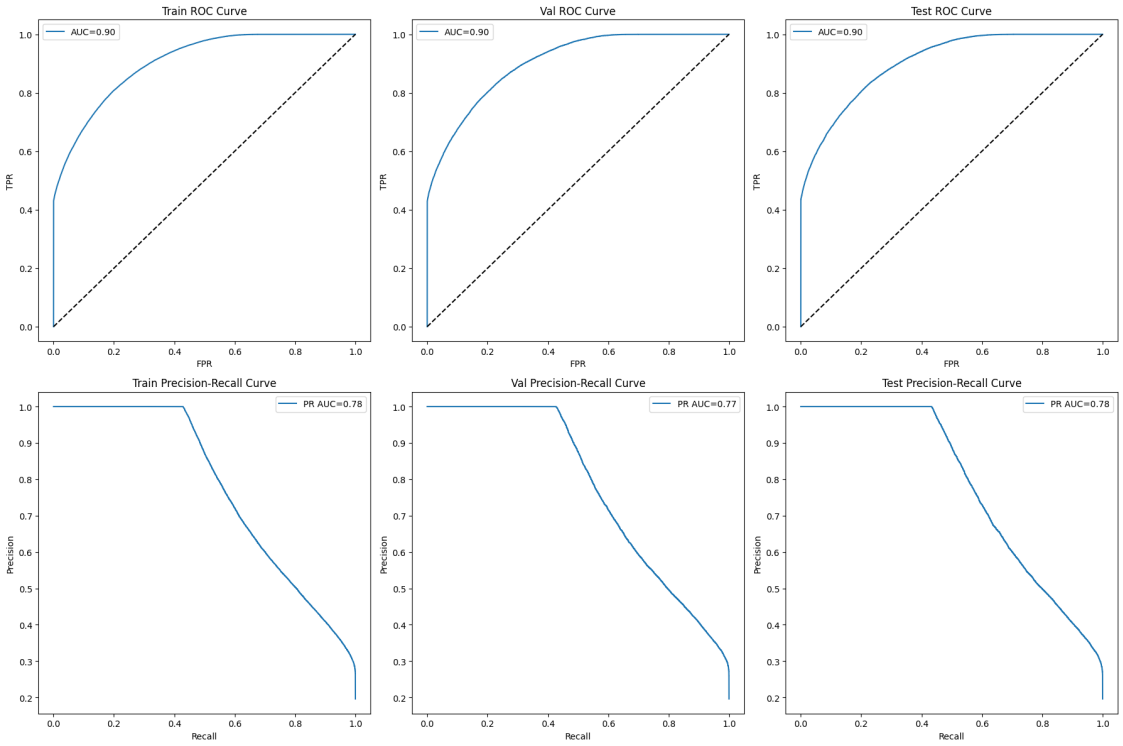
[ ]:
```
# 4. Apply threshold to create final predictions
y_pred_train = (y_pred_proba_train >= best_threshold).astype(int)
y_pred_val = (y_pred_proba_val >= best_threshold).astype(int)
y_pred_test = (y_pred_proba_test >= best_threshold).astype(int)
```

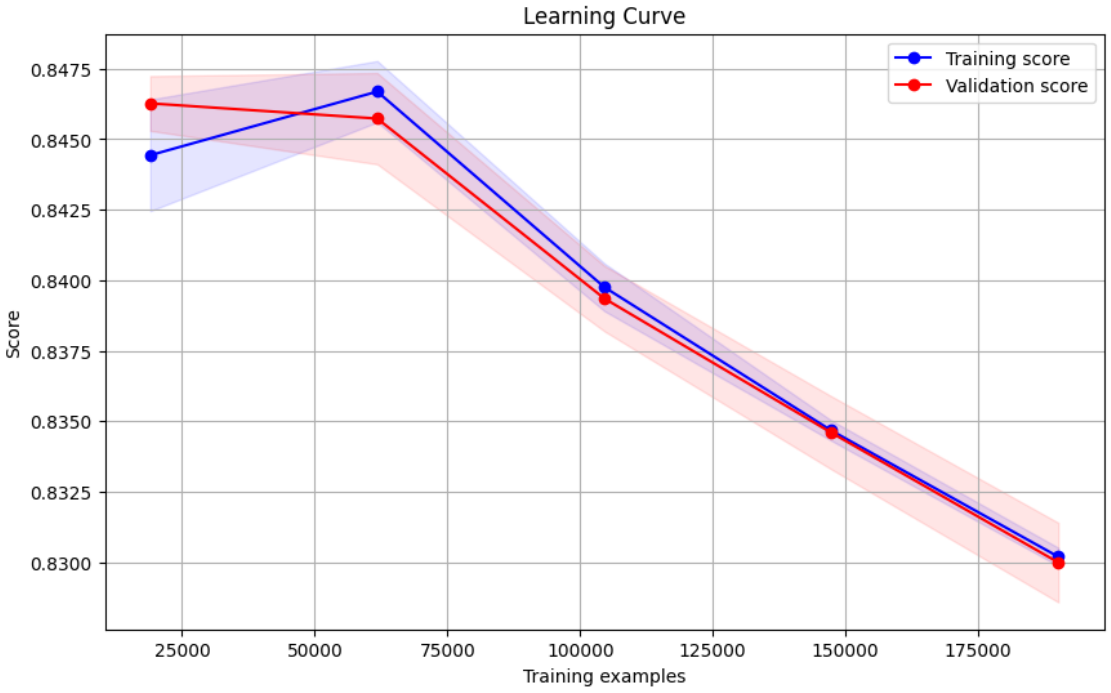
[ ]:
```
mlflow_logging_and_metric_printing(
    model=model,
    run_name="Imbalanced_Regularized_logistic_regression_after_thresholding",
    bal_type="Imbalanced",
    X_train=X_train, y_train=y_train, y_pred_train=y_pred_train,
    X_val=X_val, y_val=y_val, y_pred_val=y_pred_val,
    X_test=X_test, y_test=y_test, y_pred_test=y_pred_test,
    hyper_tuning_score=0,
    feature_names=feature_names,
    best_threshold=best_threshold,
    best_lambda=best_lambda
)
```

```
Train Metrics:
  Accuracy_train: 0.8631
  Precision_train: 0.6495
  Recall_train: 0.6561
  F1_score_train: 0.6528
  F2_score_train: 0.6548
  Roc_auc_train: 0.9043
  Pr_auc_train: 0.7769

Train Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.91      0.91    191014
           1       0.65      0.66      0.65     46604

    accuracy                           0.86    237618
   macro avg       0.78      0.78      0.78    237618
weighted avg       0.86      0.86      0.86    237618


Val Metrics:
  Accuracy_val: 0.8631
  Precision_val: 0.6510
  Recall_val: 0.6510
  F1_score_val: 0.6510
  F2_score_val: 0.6510
  Roc_auc_val: 0.9029
  Pr_auc_val: 0.7748
```

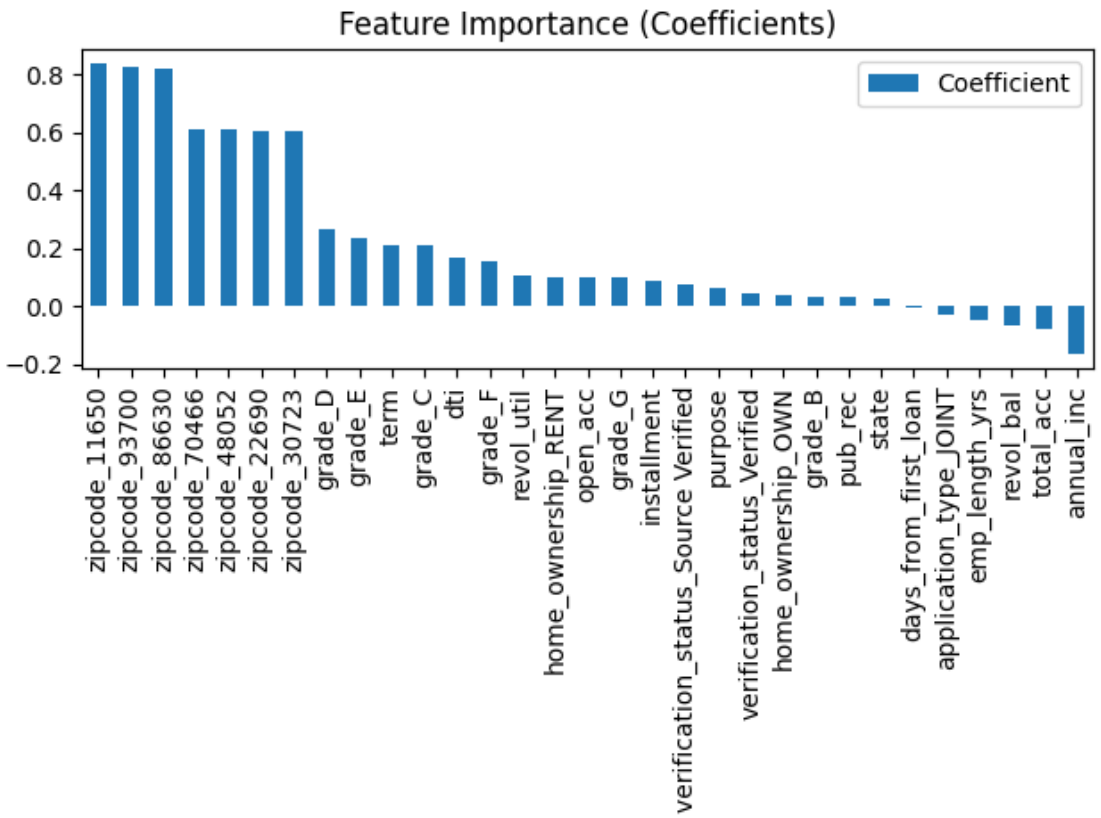
```
Val Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.91      0.91     63672
           1       0.65      0.65      0.65     15534

    accuracy                           0.86     79206
   macro avg       0.78      0.78      0.78     79206
weighted avg       0.86      0.86      0.86     79206


Test Metrics:
  Accuracy_test: 0.8649
  Precision_test: 0.6552
  Recall_test: 0.6570
  F1_score_test: 0.6561
  F2_score_test: 0.6566
  Roc_auc_test: 0.9040
  Pr_auc_test: 0.7786

Test Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.92      0.92     63671
           1       0.66      0.66      0.66     15535

    accuracy                           0.86     79206
   macro avg       0.79      0.79      0.79     79206
weighted avg       0.87      0.86      0.86     79206


Hyperparameter Tuning Best Validation Score: 0.0000
```







<Figure size 1200x800 with 0 Axes>

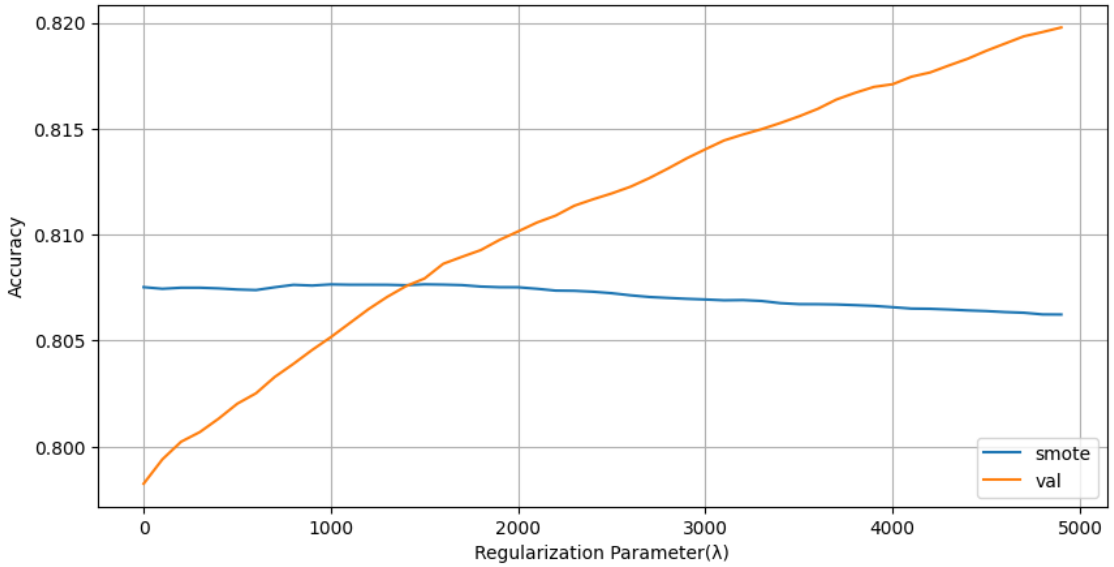



```
Model Intercept: -0.6082

2025/04/27 01:03:06 WARNING mlflow.models.model: Model logged without a
signature and input example. Please set `input_example` parameter when logging
the model to auto infer the model signature.


 MLFLOW Logging Completed.
```

17.3 Hyperparamter Tuning on Balanced data

[ ]:
```python
smote_scores = []
val_scores = []
for la in np.arange(0.01, 5000.0, 100): # range of values of Lambda
  model = LogisticRegression(C=1/la,class_weight="balanced")
  model.fit(X_smote, y_smote)
  smote_score = accuracy_score(y_smote, model.predict(X_smote))
  val_score = accuracy_score(y_val, model.predict(X_val))
  smote_scores.append(smote_score)
  val_scores.append(val_score)
```

[ ]:
```
plt.figure(figsize=(10,5))
plt.plot(list(np.arange(0.01, 5000.0, 100)), smote_scores, label="smote")
plt.plot(list(np.arange(0.01, 5000.0, 100)), val_scores, label="val")
plt.legend(loc='lower right')
plt.xlabel("Regularization Parameter( )")
plt.ylabel("Accuracy")
plt.grid()
plt.show()
```



[ ]:
```
# Find best lambda
lambdas = np.arange(0.01, 5000.0, 100)
best_lambda_index = np.argmax(val_scores)
best_lambda = lambdas[best_lambda_index]
best_val_score = val_scores[best_lambda_index]

print(f"Best Lambda ( ): {best_lambda}")
print(f"Best Validation Accuracy: {best_val_score:.4f}")
```

```
Best Lambda ( ): 4900.01
Best Validation Accuracy: 0.8198
```

17.4 Simple Logistic Regression on Balanced Data

17.4.1 Before thresholding

[ ]:
```
model = LogisticRegression(C=1/4900,class_weight="balanced")
model.fit(X_smote,y_smote)
```

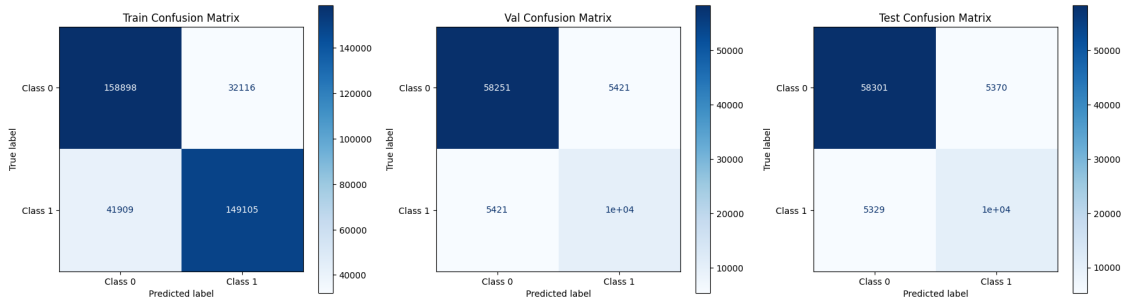
[ ]:
```
LogisticRegression(C=0.00020408163265306123, class_weight='balanced')
```

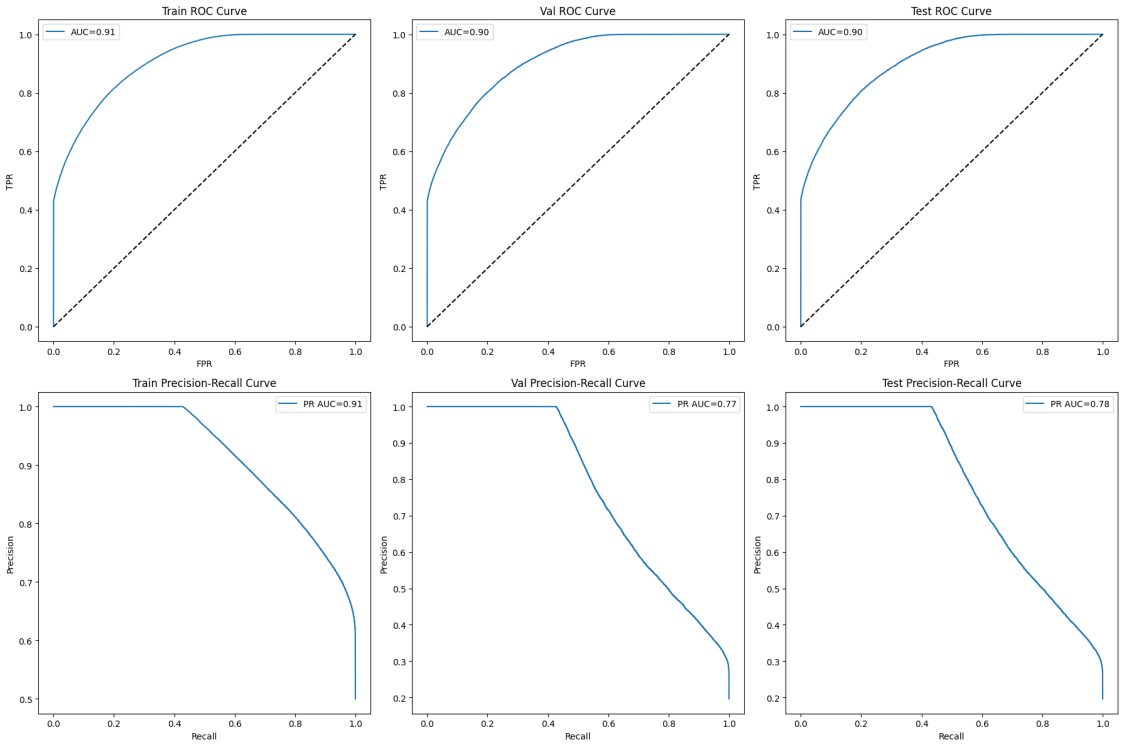
[ ]:
```
y_pred_smote = model.predict(X_smote)
y_pred_val_smote = model.predict(X_val)
y_pred_test_smote = model.predict(X_test)
```

[ ]:
```
feature_names = column_names

mlflow_logging_and_metric_printing(
    model=model,
    run_name="Balanced_Regularized_logistic_regression_before_thresholding",
    bal_type="Balanced_Smote",
    X_train=X_smote, y_train=y_smote, y_pred_train=y_pred_smote,
    X_val=X_val, y_val=y_val, y_pred_val=y_pred_val,
    X_test=X_test, y_test=y_test, y_pred_test=y_pred_test,
    hyper_tuning_score=0,
    feature_names=feature_names,
    best_lambda=best_lambda
)
```

```
Train Metrics:
  Accuracy_train: 0.8062
  Precision_train: 0.8228
  Recall_train: 0.7806
  F1_score_train: 0.8011
  F2_score_train: 0.7887
  Roc_auc_train: 0.9079
  Pr_auc_train: 0.9139

Train Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.83      0.81    191014
           1       0.82      0.78      0.80    191014

    accuracy                           0.81    382028
   macro avg       0.81      0.81      0.81    382028
weighted avg       0.81      0.81      0.81    382028


Val Metrics:
  Accuracy_val: 0.8631
  Precision_val: 0.6510
  Recall_val: 0.6510
  F1_score_val: 0.6510
  F2_score_val: 0.6510
```

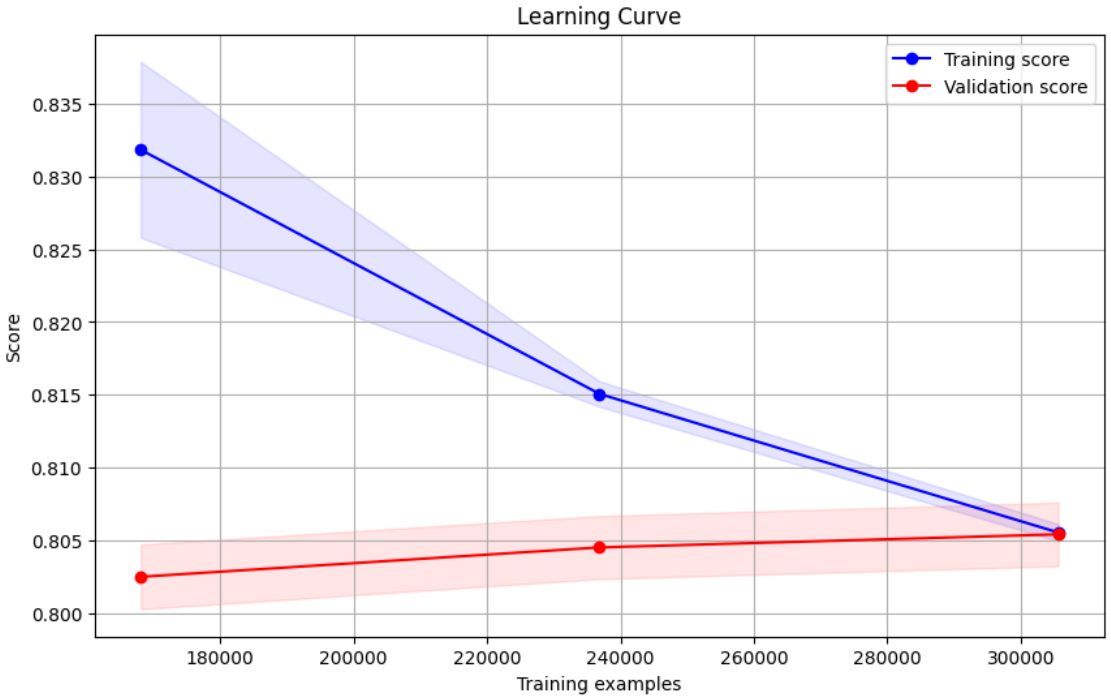
```
  Roc_auc_val: 0.9034
  Pr_auc_val: 0.7747

Val Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.91      0.91     63672
           1       0.65      0.65      0.65     15534

    accuracy                           0.86     79206
   macro avg       0.78      0.78      0.78     79206
weighted avg       0.86      0.86      0.86     79206


Test Metrics:
  Accuracy_test: 0.8649
  Precision_test: 0.6552
  Recall_test: 0.6570
  F1_score_test: 0.6561
  F2_score_test: 0.6566
  Roc_auc_test: 0.9045
  Pr_auc_test: 0.7783

Test Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.92      0.92     63671
           1       0.66      0.66      0.66     15535

    accuracy                           0.86     79206
   macro avg       0.79      0.79      0.79     79206
weighted avg       0.87      0.86      0.86     79206


Hyperparameter Tuning Best Validation Score: 0.0000
```



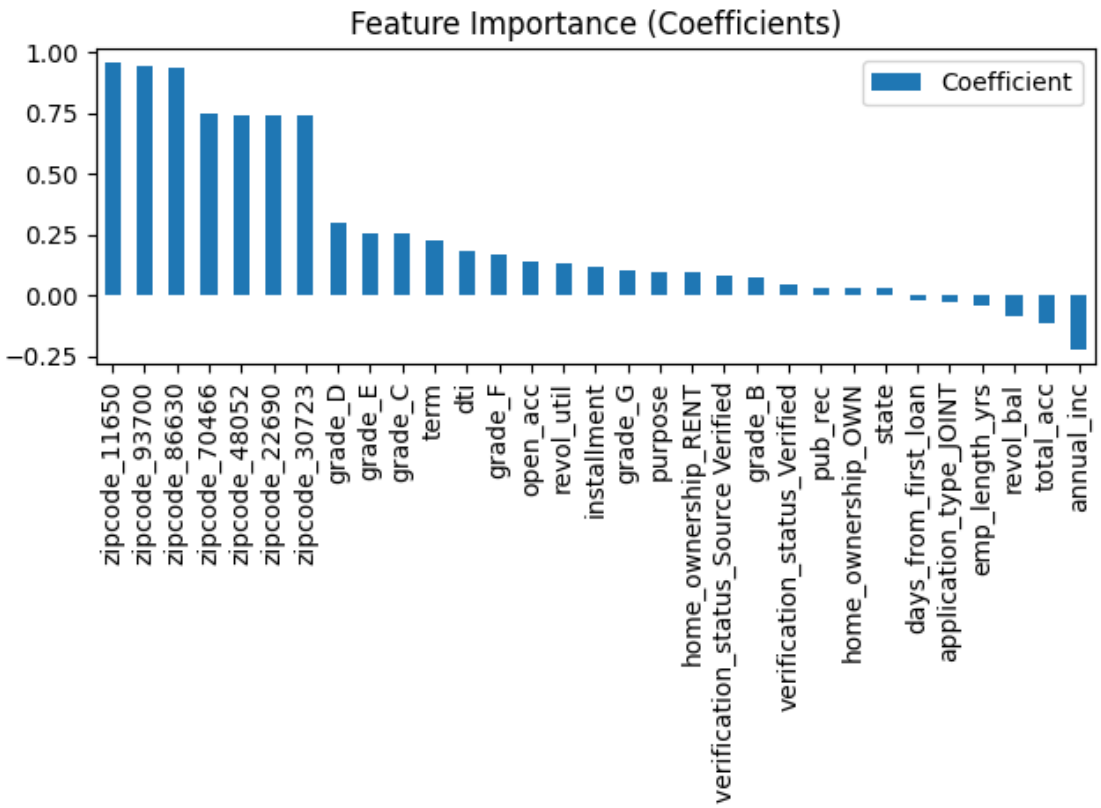



```
c:\Users\sreem\.conda\envs\ml_env\lib\site-
packages\sklearn\model_selection\_validation.py:528: FitFailedWarning:
2 fits failed out of a total of 25.
The score on these train-test partitions for these parameters will be set to
nan.
If these failures are not expected, you can try to debug them by setting
error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
2 fits failed with the following error:
Traceback (most recent call last):
  File "c:\Users\sreem\.conda\envs\ml_env\lib\site-
packages\sklearn\model_selection\_validation.py", line 866, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "c:\Users\sreem\.conda\envs\ml_env\lib\site-packages\sklearn\base.py",
line 1389, in wrapper
    return fit_method(estimator, *args, **kwargs)
  File "c:\Users\sreem\.conda\envs\ml_env\lib\site-
packages\sklearn\linear_model\_logistic.py", line 1301, in fit
    raise ValueError(
ValueError: This solver needs samples of at least 2 classes in the data, but the
data contains only one class: np.int64(0)
```

```
warnings.warn(some_fits_failed_message, FitFailedWarning)
```




```
<Figure size 1200x800 with 0 Axes>
```

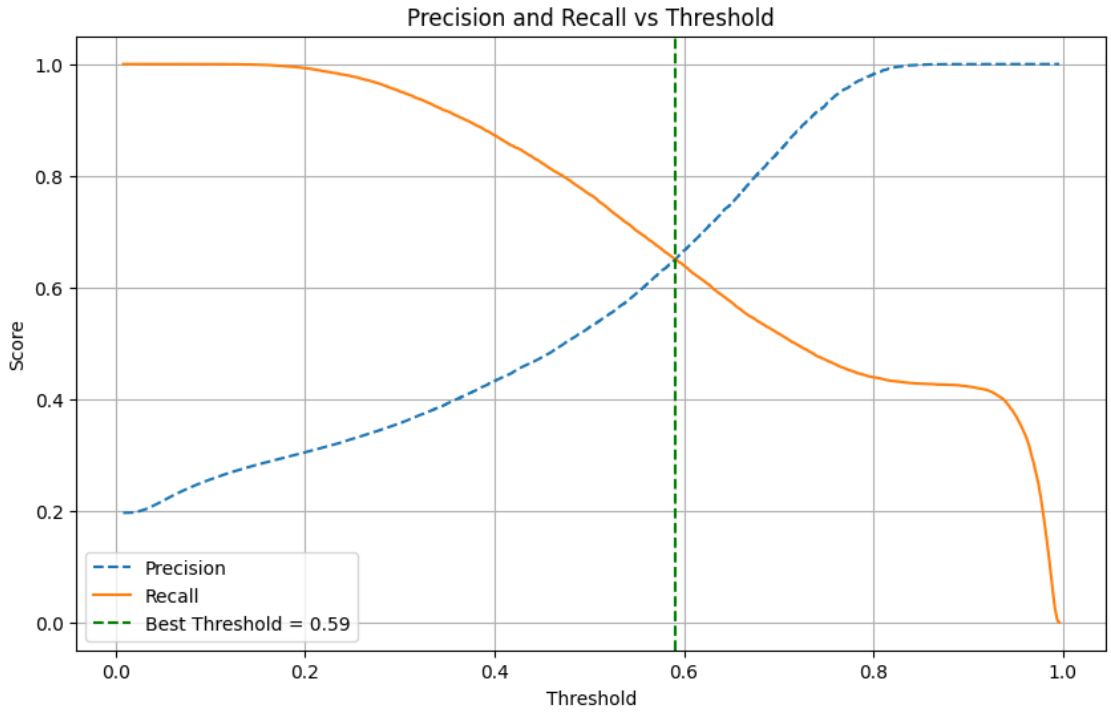



```
Model Intercept: -0.7093

2025/04/27 01:03:39 WARNING mlflow.models.model: Model logged without a
signature and input example. Please set `input_example` parameter when logging
the model to auto infer the model signature.


 MLFLOW Logging Completed.
```

17.4.2 After Thresholding

[ ]:
```
# 2. Predict probabilities
y_pred_proba_smote = model.predict_proba(X_smote)[:, 1]
y_pred_proba_val = model.predict_proba(X_val)[:, 1]
y_pred_proba_test = model.predict_proba(X_test)[:, 1]
# 3. Find Best Threshold
best_threshold = find_best_threshold(y_val, y_pred_proba_val)
```

```
Best Threshold where Precision  Recall: 0.5911
```



[ ]:
```
# After threshold tuning outside
y_pred_smote = (y_pred_proba_smote >= best_threshold).astype(int)
y_pred_val = (y_pred_proba_val >= best_threshold).astype(int)
y_pred_test = (y_pred_proba_test >= best_threshold).astype(int)
```

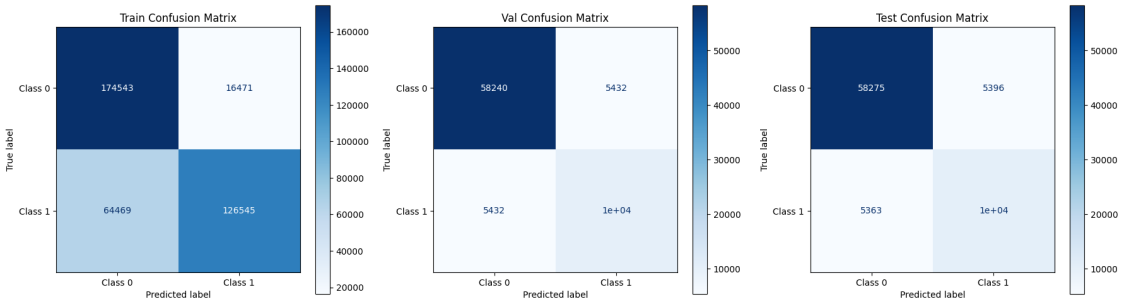
[ ]:
```
feature_names = column_names

mlflow_logging_and_metric_printing(
    model=model,
    run_name="Balanced_Regularized_logistic_regression_after_thresholding",
    bal_type="Balanced_Smote",
    X_train=X_smote, y_train=y_smote, y_pred_train=y_pred_smote,
    X_val=X_val, y_val=y_val, y_pred_val=y_pred_val,
    X_test=X_test, y_test=y_test, y_pred_test=y_pred_test,
    hyper_tuning_score=0,
    feature_names=feature_names,
    best_threshold=best_threshold,
    best_lambda=best_lambda
)
```

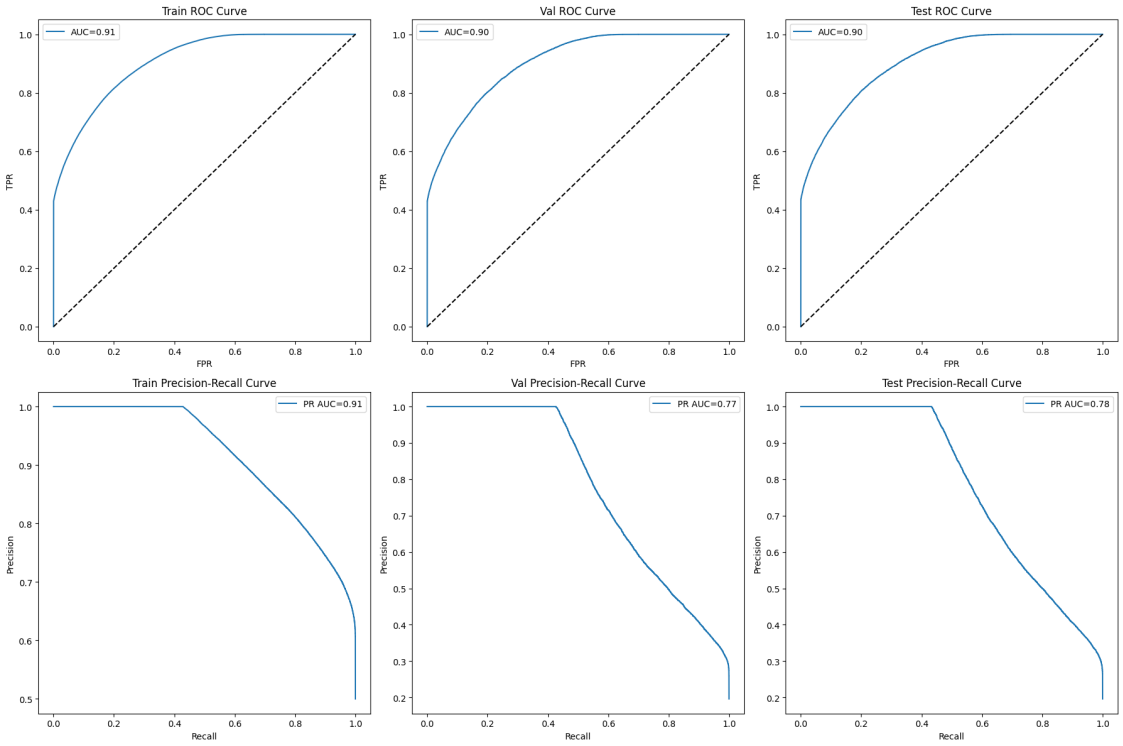
```
Train Metrics:
  Accuracy_train: 0.7881
  Precision_train: 0.8848
```

```
  Recall_train: 0.6625
  F1_score_train: 0.7577
  F2_score_train: 0.6975
  Roc_auc_train: 0.9079
  Pr_auc_train: 0.9139

Train Classification Report:
              precision    recall  f1-score   support

           0       0.73      0.91      0.81    191014
           1       0.88      0.66      0.76    191014

    accuracy                           0.79    382028
   macro avg       0.81      0.79      0.78    382028
weighted avg       0.81      0.79      0.78    382028


Val Metrics:
  Accuracy_val: 0.8628
  Precision_val: 0.6503
  Recall_val: 0.6503
  F1_score_val: 0.6503
  F2_score_val: 0.6503
  Roc_auc_val: 0.9034
  Pr_auc_val: 0.7747

Val Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.91      0.91     63672
           1       0.65      0.65      0.65     15534

    accuracy                           0.86     79206
   macro avg       0.78      0.78      0.78     79206
weighted avg       0.86      0.86      0.86     79206


Test Metrics:
  Accuracy_test: 0.8642
  Precision_test: 0.6534
  Recall_test: 0.6548
  F1_score_test: 0.6541
  F2_score_test: 0.6545
  Roc_auc_test: 0.9045
  Pr_auc_test: 0.7783

Test Classification Report:
              precision    recall  f1-score   support
```

```
              0       0.92      0.92      0.92     63671
              1       0.65      0.65      0.65     15535

       accuracy                           0.86     79206
      macro avg       0.78      0.79      0.78     79206
   weighted avg       0.86      0.86      0.86     79206
```

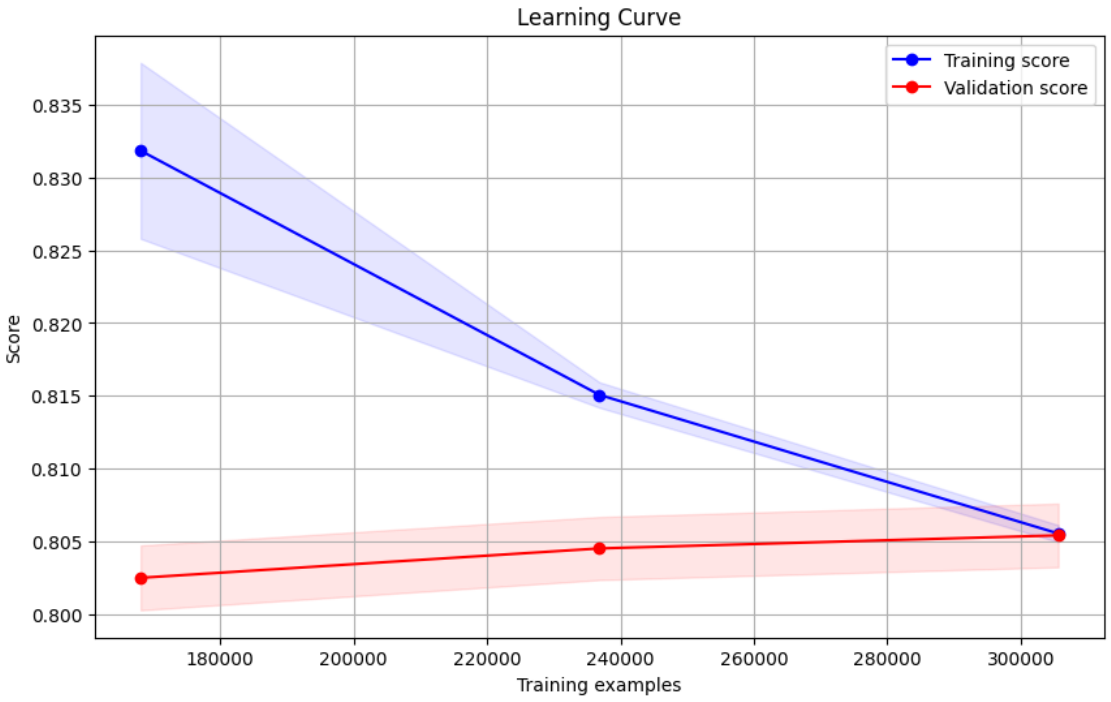
```
Hyperparameter Tuning Best Validation Score: 0.0000
```



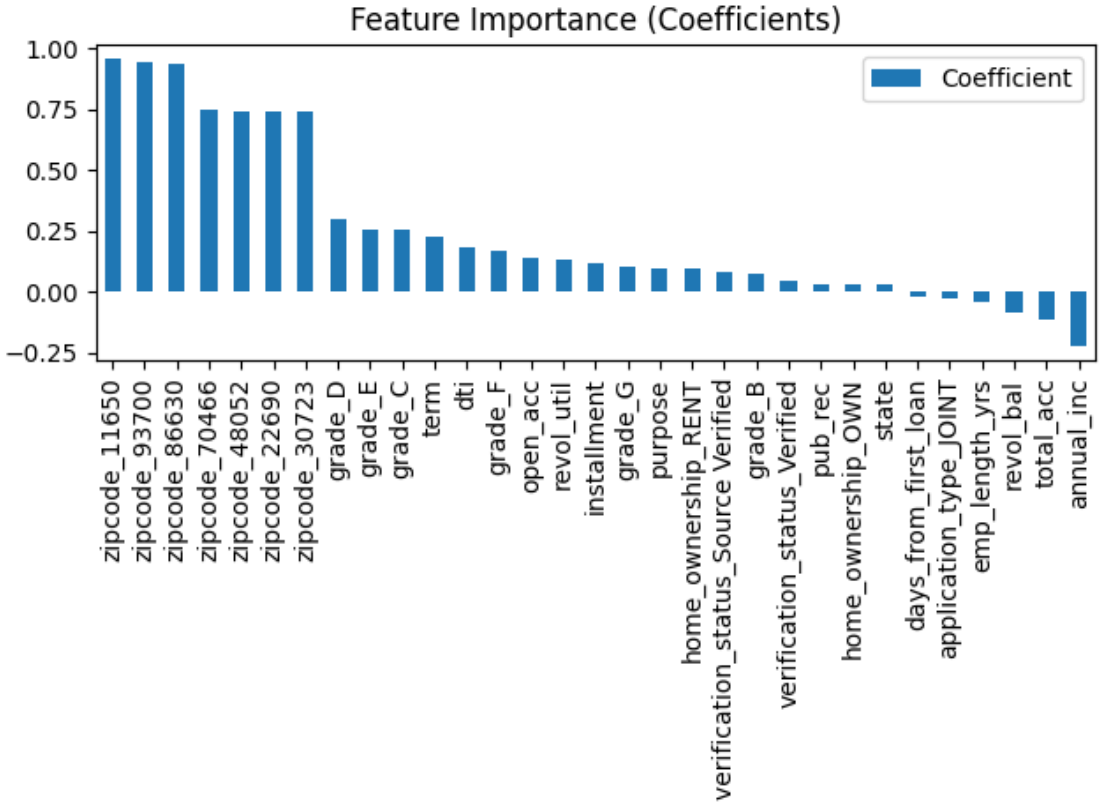



```
c:\Users\sreem\.conda\envs\ml_env\lib\site-
```

```
packages\sklearn\model_selection\_validation.py:528: FitFailedWarning:
2 fits failed out of a total of 25.
The score on these train-test partitions for these parameters will be set to
nan.
If these failures are not expected, you can try to debug them by setting
error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
2 fits failed with the following error:
Traceback (most recent call last):
  File "c:\Users\sreem\.conda\envs\ml_env\lib\site-
packages\sklearn\model_selection\_validation.py", line 866, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "c:\Users\sreem\.conda\envs\ml_env\lib\site-packages\sklearn\base.py",
line 1389, in wrapper
    return fit_method(estimator, *args, **kwargs)
  File "c:\Users\sreem\.conda\envs\ml_env\lib\site-
packages\sklearn\linear_model\_logistic.py", line 1301, in fit
    raise ValueError(
ValueError: This solver needs samples of at least 2 classes in the data, but the
data contains only one class: np.int64(0)

  warnings.warn(some_fits_failed_message, FitFailedWarning)
```



<Figure size 1200x800 with 0 Axes>



Model Intercept: -0.7093

2025/04/27 01:03:50 WARNING mlflow.models.model: Model logged without a signature and input example. Please set `input_example` parameter when logging the model to auto infer the model signature.

MLFLOW Logging Completed.

18 Results Comparison

19 Result Comparision

Model Variant	ROC AUC	Precision	Recall	F1 Score	Accuracy
Imbalanced Logistic (No Threshold Adj.)	0.9061	0.9451	0.4665	0.6247	0.89
Imbalanced Logistic (Threshold Tuned)	0.9061	0.6543	0.6565	0.6554	0.86
Balanced (SMOTE) Logistic (No Threshold Adj.)	0.9040	0.5420	0.7542	0.6307	0.86
Balanced (SMOTE) Logistic (Threshold Tuned)	0.9040	0.6552	0.6570	0.6561	0.86
Imbalanced Regularized Logistic (No Thr. Adj.)	0.9029	0.5415	0.7534	0.6301	0.83
Imbalanced Regularized Logistic (Threshold Tuned)	0.9057	0.6520	0.6556	0.6538	0.86
Balanced Reg. (SMOTE) Logistic (No Thr. Adj.)	0.9040	0.54	0.75	0.63	0.86
Balanced Reg. (SMOTE) Logistic (Threshold Tuned)	0.9045	0.6534	0.6548	0.6541	0.86

20 Trade-off questions:

20.0.1 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.

Real Defaulters means True positives. So to give more importance to TP and Reduce False Positives. It is indicating that to improve the precision score. That means more weightage should be given for Precision score

f beta score with Beta less than 1 should be selected. Beta can be taken as hyper parameter to find best Beta score. But On decrease of Beta, Recall score will decrease. We can take Specificity as metric, Where TN increases and FP decreases

If False Negatives increases, We lose out an oppurtunity to finance more individuals. SO to minimise the FN, We have use Recall or sensitivity or f Beta score with Beta greater than 1 or False Negative Rate

20.0.2 Since NPA (non-performing asset) is a real problem in this industry, it's important we play safe and shouldn't disburse loans to anyone.

To play safe, False positive should be minimised. So we have to use F Beta score with Beta less than 1 or Specificity or Recall score as hyper parameter tuning metric

21 Actionable Insights and Recommendations

1. **Focus on Recall to Reduce Default Risk:** The initial model had very high precision (~94.5%) but low recall (~46.7%), meaning many defaulters were not being caught. Tuning the classification threshold or balancing the dataset dramatically increased recall (to ~65%) at the expense of some precision. From a lender's perspective, it's more critical to catch as many potential defaulters as possible – **prioritize a model configuration with higher recall**, even if precision drops. This will flag more high-risk loans and reduce losses from undetected defaulters, albeit with a slightly higher false-alarm rate (denied loans to some safe customers).
2. **Threshold Tuning and Data Balancing:** Simply oversampling the minority class (defaults) or using class-weighting without adjusting the decision threshold led to recall ~75% but precision ~54%. On the other hand, using the optimal threshold where precision recall yields a more balanced outcome (precision recall 65%). It is recommended to **combine class rebalancing with threshold tuning** to achieve a balance that aligns with business goals. This approach improved the F1 score and ensures neither type of error (false approval or false denial) dominates.
3. **Regularization Effects:** Introducing L2 regularization (with an optimal found via hyperparameter tuning) did not significantly degrade model discrimination. The ROC AUC remained around ~0.904–0.906 across all variants. Regularization mainly helps to simplify the model and potentially improve generalization. We **recommend keeping regularization (L2) in the model** to avoid overfitting, while noting that it had minimal impact on the primary performance metrics in this case (due to the large dataset).
4. **Key Predictors – Leverage Feature Insights:** The analysis indicates that certain features have strong influence on loan repayment. In particular, **interest rate (loan**

grade/subgrade), **loan amount and installment size**, **debt-to-income ratio (DTI)**, and **revolving credit utilization** emerged as important factors (high coefficient magnitudes in the logistic model). For instance, higher interest rates (worse grades) correlate with higher default probability, and larger loan amounts with big installments can strain borrowers. It is recommended to **closely monitor these factors in credit risk assessments**. Borrowers with high DTI or credit utilization should be scrutinized or offered smaller loans to mitigate risk.

5. **Customer Profile and Policy Adjustments:** About **19.6%** of the loans in the data ended up charged off (defaulted). Many of these defaults could potentially be averted by tightening lending criteria. **Stricter approval for high-risk groups** (e.g., borrowers with low grades, high DTI, or unverified income) can improve the portfolio's overall health. The bank might consider policy interventions like requiring co-signers or collateral for loans that fall into a higher-risk category identified by the model.
6. **Ongoing Model Monitoring:** Given that roughly **80%** of loans are fully paid, the model will be exposed to class imbalance in production. Continuously track the precision-recall balance. If the default rate changes or if model performance drifts, **re-tune the threshold or retrain with updated data**. Monitoring the model's predictive power (ROC AUC ~0.90 as a benchmark) over time will ensure it remains effective. Regular recalibration can align the model with current economic conditions or shifts in borrower behavior (e.g., economic downturns might increase default rates and warrant a threshold adjustment to maintain recall).

22 Questions and Answers

1. **What percentage of customers have fully paid their Loan Amount?** – Approximately **80.4%** of customers fully paid their loans (318,357 out of 396,030 total loans).
2. **Comment about the correlation between Loan Amount and Installment features.** – **Loan amount and installment are highly positively correlated.** Borrowers with larger loan amounts have proportionally higher monthly installments; in fact, the installment is essentially the EMI calculated from the loan amount, term, and interest rate.
3. **The majority of people have home ownership as _____.** – The majority of borrowers (about **50%**) have **"MORTGAGE"** as their home ownership status, followed by those who RENT (~40%). This indicates most loan applicants are paying a mortgage on a home.
4. **People with grades 'A' are more likely to fully pay their loan. (T/F)** – **True.** Grade 'A' loans (the highest quality) show a higher proportion of fully paid loans compared to charged-off loans. Higher grades correspond to lower default risk, so 'A' grade borrowers are indeed more likely to pay off their loans in full.
5. **Name the top 2 afforded job titles.** – The two most common employment titles among borrowers are **"Unknown"** (missing/unspecified job title, appearing over 17k times) and **"Teacher"** (the most frequent specified job, ~4.3k borrowers). ("Manager" is a close third with ~4.2k borrowers.)
6. **Thinking from a bank's perspective, which metric should our primary focus be on?** – From the bank's perspective, **Recall** is paramount. The bank would want to catch as many defaulters as possible (high recall), to avoid lending to high-risk customers who won't repay. Focusing on recall (with an acceptable precision trade-off) helps minimize the number of defaulting loans.

7. **How does the gap in precision and recall affect the bank?** – A large gap indicates an imbalance in errors:
   - If **precision recall** (as in the initial model), the bank approves most loans and rarely mislabels a good customer as risky, but it **misses many actual defaulters**. This leads to more bad loans slipping through (losses increase).
   - If **recall precision**, the bank catches most defaulters but at the cost of **denying many creditworthy customers** (false positives). This can reduce revenue and customer goodwill.
   - We need to find a balance where recall is high enough to flag risky loans, while precision is still reasonable to not turn away too many good customers.
8. **Which were the features that heavily affected the outcome?** – The model coefficients and data exploration suggest **interest rate (which corresponds to loan grade/subgrade)**, **installment (linked to loan amount)**, **debt-to-income ratio (dti)**, **revolving credit utilization (revol_util)**, and **length of credit history** are influential features. For example, borrowers with higher interest rates, larger loans (hence higher installments), or high dti/revol_util had higher chances of defaulting, all else equal. These features drove the model's predictions and are key for underwriting.
9. **Will the results be affected by geographical location? (Yes/No)** – **No.** The model did not find a strong signal based on geography. Loans were issued across many states, but no location stood out as having notably different default behavior in the analysis. Thus, the predictive results and conclusions are **not significantly affected by the borrower's state/region**. (The top states by loan count were California, New York, etc., but default rates in those states were in line with the national trend.)

[ ]: