Case Study: LoanTap: Logistic Regression

1.0 Defining the Problem Statement

1.1 About the LoanTap Organisation:

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

1.2 Problem Statement

Objective: - Develop a robust credit underwriting model for personal loans. - Accurately predict the creditworthiness of individual loan applicants. - Determine optimal repayment terms (e.g., loan amount, interest rate, tenure) for approved applications.

Scope: - This case study focuses exclusively on the underwriting process for personal loans within the LoanTap ecosystem. - The model will utilize a given set of applicant attributes to make credit decisions.

Key Deliverables: - A predictive model that accurately classifies applicants as "creditworthy" or "not creditworthy." - Business recommendations to optimize the lending process and minimize risk.

2.0 Importing the Libraries and Loading the Dataset

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

```
# !gdown https://drive.google.com/uc?id=1ZPYj7CZCfxntE8p2Lze_4Q04MyEOy6_d -O
"LoanTap_dataset.csv"
```

```
# df = pd.read_csv("/content/LoanTap_dataset.csv")
df = pd.read_csv("/content/LoanTap_logistic_regression.csv")
```

Sampling the dataset

```
# Code to display all the rows and column
pd.set_option('display.max_columns', None)
```

```
df.sample(5, random_state = 42)
```

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp
362323	14000.0	60 months	14.49	329.33	C	C4	mental health tech	10+
220444	6050.0	36 months	16.29	213.57	D	D2	teller	2 ye
345899	20775.0	36 months	18.24	753.57	D	D5	Business Analyst	2 ye
93811	6000.0	36 months	13.99	205.04	C	C4	Outreach and Enrollment	1 ye
182096	17450.0	36 months	13.11	588.89	B	B4	Pinellas county schools	8 ye

3.0 Exploratiry Data Analysis

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 396030 entries, 0 to 396029
Data columns (total 27 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   loan_amnt             396030 non-null  float64
 1   term                  396030 non-null  object
 2   int_rate              396030 non-null  float64
 3   installment           396030 non-null  float64
 4   grade                 396030 non-null  object
 5   sub_grade             396030 non-null  object
 6   emp_title             373103 non-null  object
 7   emp_length            377729 non-null  object
 8   home_ownership        396030 non-null  object
 9   annual_inc            396030 non-null  float64
 10  verification_status   396030 non-null  object
 11  issue_d               396030 non-null  object
 12  loan_status           396030 non-null  object
 13  purpose               396030 non-null  object
 14  title                 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
```

```python
df.shape
print(f'Total number of rows    : {df.shape[0]}')
print(f'Total number of columns : {df.shape[1]}')
```

```
Total number of rows    : 396030
Total number of columns : 27
```

Key Observations

- **Dataset Dimensions:** The dataset comprises 27 rows and 396,030 columns.
- **Data Quality:** Some columns contain null values.
- **Feature Selection:** 26 variables are identified as potential predictors for the target variable "loan_status," based on the business context.

Next Steps

- **Data Quality Assessment:**
  - Calculate the percentage of missing values in each column.
  - Determine the overall percentage of missing values within the entire dataset.
  - This analysis will guide the appropriate strategies for handling null values (e.g., imputation, removal).
- **Feature Engineering:**
  - Implement feature engineering techniques relevant to the specific business requirements.
    - This may involve:
      - Creating new features from existing ones.
      - Transforming existing features (e.g., scaling, binning, encoding categorical variables).
      - Selecting the most relevant features for the predictive model.

```python
# Function to display, only the columns with null values along with missing
values count.
def missing_values():
  missing_values = df.isnull().sum()
  missing_values = missing_values[missing_values > 0].rename('Missing Values')
  return missing_values

missing_values()
```

	Missing Values
emp_title	22927
emp_length	18301
title	1756
revol_util	276
mort_acc	37795
pub_rec_bankruptcies	535

```python
# Function to calculate missing value percentages across each columns.
def missing_value_perc():
  missing_value_perc = np.round(100*(df.isnull().sum()/df.shape[0]),2)
  missing_value_perc = missing_value_perc[missing_value_perc > 0].rename('Missing
Values Percentage')
  return missing_value_perc

missing_value_perc()
```

	Missing Values Percentage
emp_title	5.79
emp_length	4.62
title	0.44
revol_util	0.07
mort_acc	9.54
pub_rec_bankruptcies	0.14

Key Observations

- **Missing Data:** The column "mort_acc" exhibits the highest rate of missing values, approaching 10%.
- **Categorical Features:** The dataset includes six categorical columns, notably "emp_title" and "title."

Next Steps

- **"mort_acc" Analysis:**

  - Given its potential significance in underwriting, a thorough investigation of the "mort_acc" column is crucial.
  - This may involve:
    * Imputation strategies to handle missing values.
    * Exploratory analysis to understand its distribution and relationship with** other variables.

- **Feature Importance:**
  - Conduct a comprehensive analysis of the relationship between each independent feature and the target variable ("loan_status").
  - This will help identify the most influential predictors and guide feature selection for subsequent modeling.

```
# Identifying unique values
for i in df.select_dtypes(include='object').columns:
  print(df[i].value_counts(dropna=False).sort_index())
  print("***"*10)
```

```
term
36 months    302005
60 months     94025
Name: count, dtype: int64
******************************
grade
A     64187
B    116018
C    105987
D     63524
E     31488
F     11772
G      3054
Name: count, dtype: int64
******************************
sub_grade
A1     9729
A2     9567
A3    10576
A4    15789
A5    18526
B1    19182
B2    22495
B3    26655
B4    25601
B5    22085
C1    23662
C2    22580
C3    21221
```

```
C4    20280
C5    18244
D1    15993
D2    13951
D3    12223
D4    11657
D5     9700
E1     7917
E2     7431
E3     6207
E4     5361
E5     4572
F1     3536
F2     2766
F3     2286
F4     1787
F5     1397
G1     1058
G2      754
G3      552
G4      374
G5      316
Name: count, dtype: int64
******************************
emp_title
       NSA Industries llc              1
   Fibro Source                        1
   Long Ilsand College Hospital        1
   mortgage banker                     1
  Credit rev specialist                1
                                    ...
zs backroom                            1
zueck transportation                   1
zulily                                 1
License Compliance Investigator       1
NaN                                22927
Name: count, Length: 173106, dtype: int64
******************************
emp_length
1 year        25882
10+ years    126041
2 years       35827
3 years       31665
4 years       23952
5 years       26495
6 years       20841
7 years       20819
8 years       19168
```

```
9 years       15314
< 1 year      31725
NaN           18301
Name: count, dtype: int64
******************************
home_ownership
ANY              3
MORTGAGE    198348
NONE            31
OTHER          112
OWN          37746
RENT        159790
Name: count, dtype: int64
******************************
verification_status
Not Verified       125082
Source Verified    131385
Verified           139563
Name: count, dtype: int64
******************************
issue_d
Apr-2008     122
Apr-2009     227
Apr-2010     648
Apr-2011    1231
Apr-2012    2508
            ...
Sep-2012    4707
Sep-2013    9179
Sep-2014    4293
Sep-2015    5419
Sep-2016    1059
Name: count, Length: 115, dtype: int64
******************************
loan_status
Charged Off     77673
Fully Paid     318357
Name: count, dtype: int64
******************************
purpose
car                     4697
credit_card            83019
debt_consolidation    234507
educational              257
home_improvement       24030
house                   2201
major_purchase          8790
medical                 4196
```

```
moving                  2854
other                  21185
renewable_energy         329
small_business          5701
vacation                2452
wedding                 1812
Name: count, dtype: int64
******************************
title
\tcredit_card                   1
\tdebt_consolidation            3
\tother                         4
\tsmall_business                2
      debt consolidation        1
                               ...
zonball Loan                    1
zxcvb                           1
~Life Reorganization~           1
~Summer Fun~                    1
NaN                          1756
Name: count, Length: 48817, dtype: int64
******************************
earliest_cr_line
Apr-1955      2
Apr-1958      1
Apr-1960      1
Apr-1961      2
Apr-1962      3
           ...
Sep-2009    391
Sep-2010    346
Sep-2011    209
Sep-2012     54
Sep-2013      3
Name: count, Length: 684, dtype: int64
******************************
initial_list_status
f    238066
w    157964
Name: count, dtype: int64
******************************
application_type
DIRECT_PAY       286
INDIVIDUAL    395319
JOINT            425
Name: count, dtype: int64
******************************
address
```

```
000 Adam Station Apt. 329\r\nAshleyberg, AZ 22690    1
000 Adrian Cliffs\r\nRandyton, LA 22690              1
000 Alexandria Street\r\nPort Richard, FL 22690      1
000 Amber Court\r\nLake Pamelatown, IN 00813         1
000 Amy Pines Suite 498\r\nSouth Susan, ND 22690     1
                                                    ..
Unit 9995 Box 6277\r\nDPO AE 48052                   1
Unit 9995 Box 8360\r\nDPO AP 00813                   1
Unit 9996 Box 9255\r\nDPO AP 05113                   1
Unit 9997 Box 3228\r\nDPO AA 11650                   1
Unit 9997 Box 3834\r\nDPO AP 86630                   1
Name: count, Length: 393700, dtype: int64
******************************
```

```
df.describe().T
```

	count	mean	std	min	25%	50%	75%	ma
loan_amnt	396030.0	14113.888089	8357.441341	500.00	8000.00	12000.00	20000.00	400
int_rate	396030.0	13.639400	4.472157	5.32	10.49	13.33	16.49	30.
installment	396030.0	431.849698	250.727790	16.08	250.33	375.43	567.30	153
annual_inc	396030.0	74203.175798	61637.621158	0.00	45000.00	64000.00	90000.00	870
dti	396030.0	17.379514	18.019092	0.00	11.28	16.91	22.98	999
open_acc	396030.0	11.311153	5.137649	0.00	8.00	10.00	14.00	90.
pub_rec	396030.0	0.178191	0.530671	0.00	0.00	0.00	0.00	86.
revol_bal	396030.0	15844.539853	20591.836109	0.00	6025.00	11181.00	19620.00	174
revol_util	395754.0	53.791749	24.452193	0.00	35.80	54.80	72.90	892
total_acc	396030.0	25.414744	11.886991	2.00	17.00	24.00	32.00	151
mort_acc	358235.0	1.813991	2.147930	0.00	0.00	1.00	3.00	34.
pub_rec_bankruptcies	395495.0	0.121648	0.356174	0.00	0.00	0.00	0.00	8.0

```
df.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

	count	unique	top	freq
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 Johnson\r\nFPO AE 48052	8

```
df.select_dtypes(include='number').skew()
```

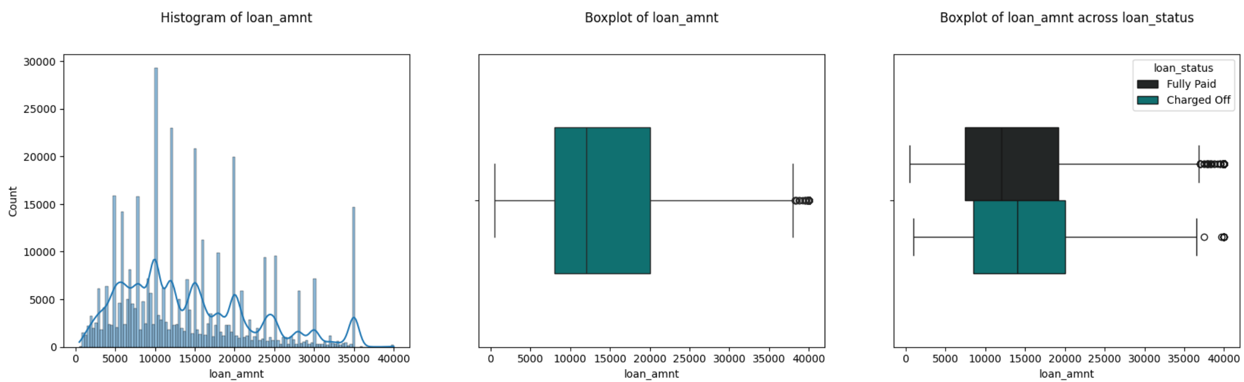
	0
loan_amnt	0.777285
int_rate	0.420669
installment	0.983598
annual_inc	41.042725
dti	431.051225
open_acc	1.213019
pub_rec	16.576564
revol_bal	11.727515
revol_util	-0.071778
total_acc	0.864328
mort_acc	1.600132
pub_rec_bankruptcies	3.423440

4.0 Graphical Analysis

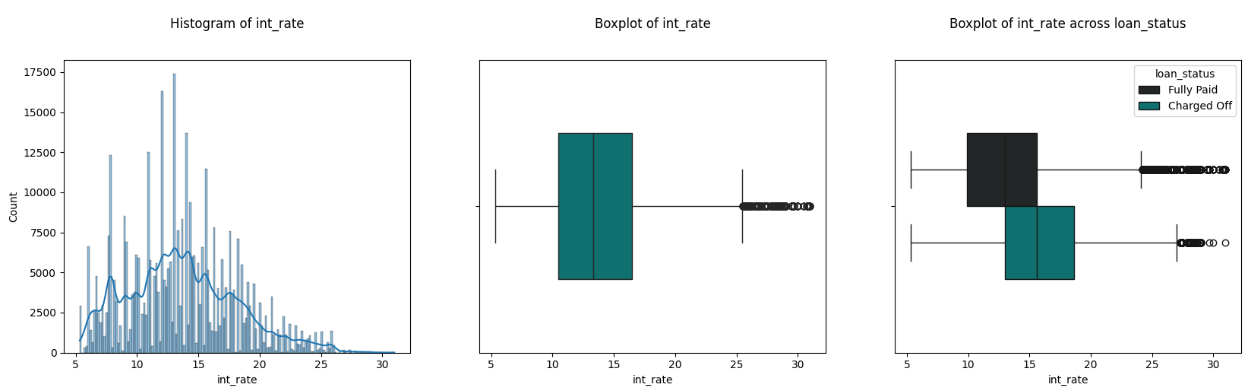
4.1 Analysis of Numerical Columns

```
numerical =['loan_amnt', 'int_rate', 'installment', 'annual_inc', 'dti',
'open_acc', 'revol_bal', 'revol_util', 'total_acc']

for i in numerical:
  fig, axes = plt.subplots(1,3, figsize = (20,5))
  sns.histplot(data = df, x= df[i], kde = True, ax = axes[0])
  axes[0].set_title(f"Histogram of {i}", pad = 30)
  # for j in axes[0].patches:
  #   values = j.get_height()
  #   percentage = 100 * values / len(df)
  #   axes[0].annotate(f'{values}\n({percentage:.1f}%)', (j.get_x() +
  #   j.get_width()/2, j.get_height()+3), ha='center', va='bottom', fontsize=10)
  sns.boxplot(data = df, x = df[i], ax = axes[1], width = 0.5, color='teal')
  axes[1].set_title(f'Boxplot of {i}', pad = 30)
  sns.boxplot(data = df, x = df[i], ax = axes[2], width = 0.5,  color='teal', hue
= 'loan_status')
```

```
axes[2].set_title(f'Boxplot of {i} across loan_status', pad = 30)
plt.show()
tab_col = pd.DataFrame(df[i].describe()).reset_index()
tab_col.columns = ['Stat', 'Value']
display(tab_col)
```

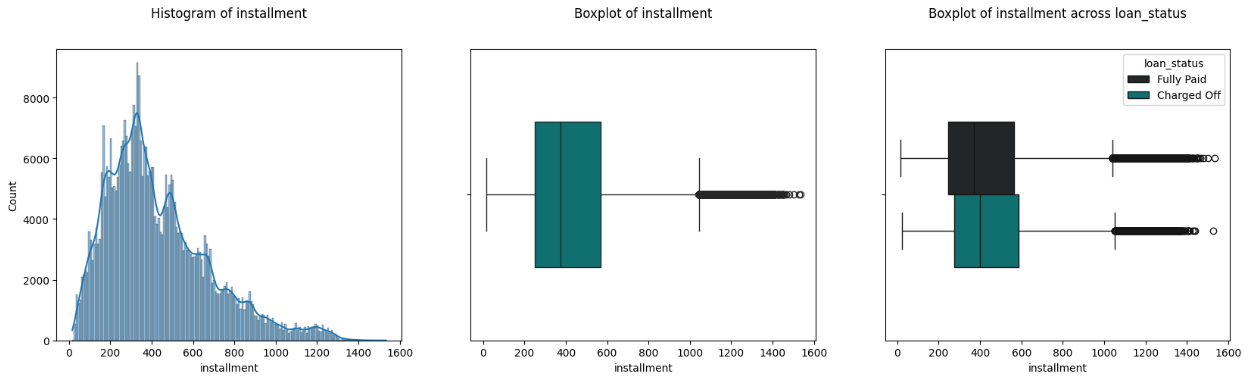


	Stat	Value
0	count	396030.000000
1	mean	14113.888089
2	std	8357.441341
3	min	500.000000
4	25%	8000.000000
5	50%	12000.000000
6	75%	20000.000000
7	max	40000.000000

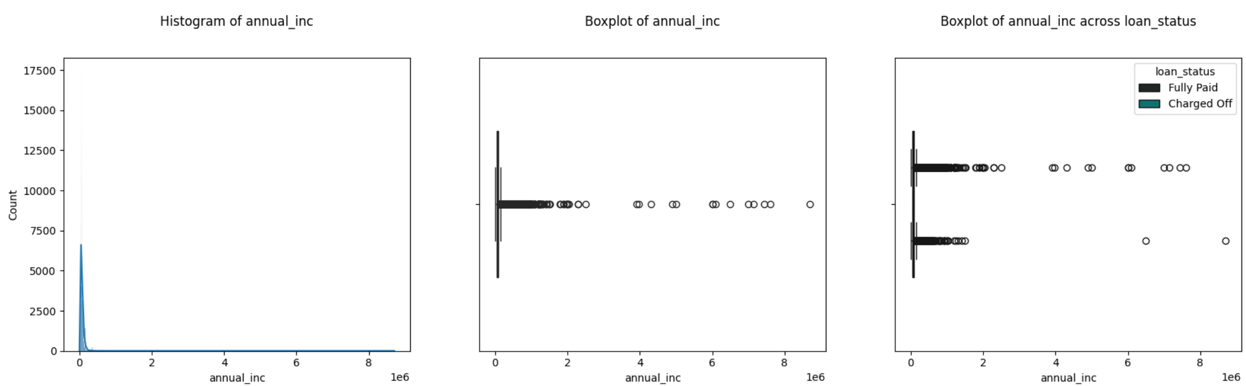


	Stat	Value
0	count	396030.000000
1	mean	13.639400
2	std	4.472157
3	min	5.320000
4	25%	10.490000

	Stat	Value
5	50%	13.330000
6	75%	16.490000
7	max	30.990000

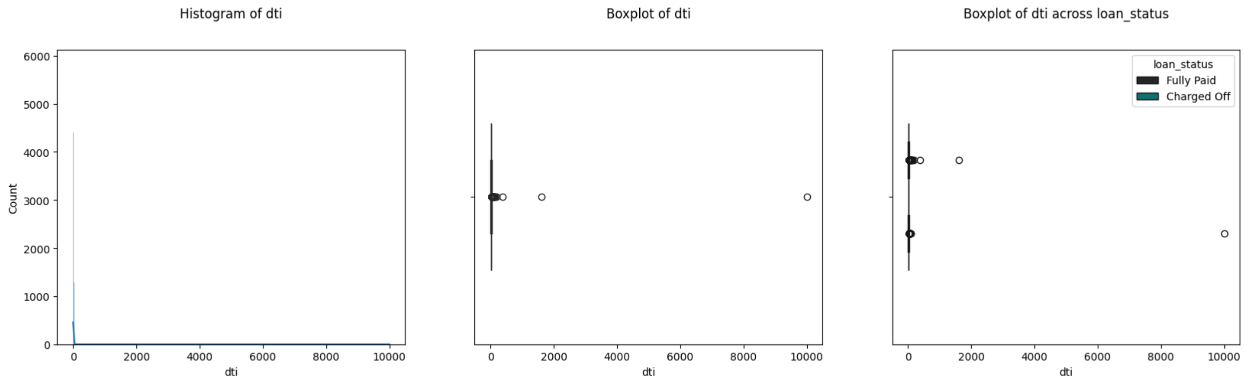


	Stat	Value
0	count	396030.000000
1	mean	431.849698
2	std	250.727790
3	min	16.080000
4	25%	250.330000
5	50%	375.430000
6	75%	567.300000
7	max	1533.810000

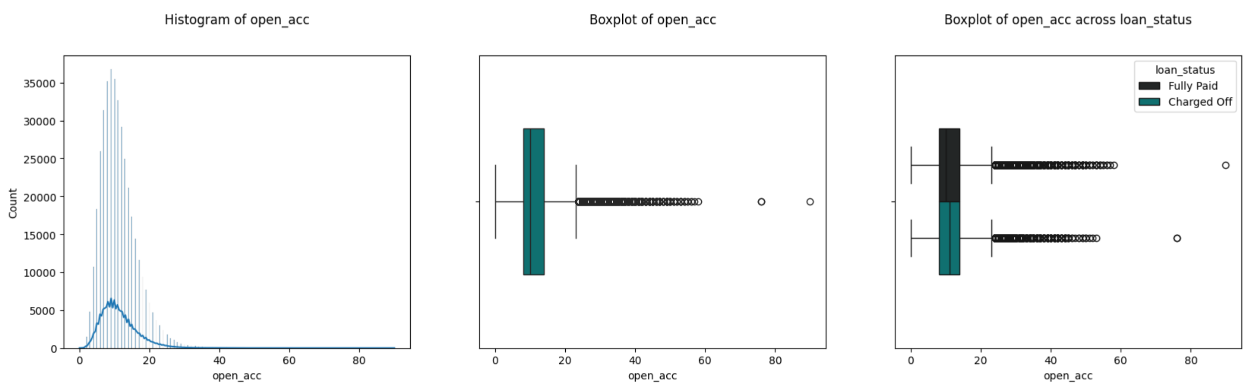


	Stat	Value
0	count	3.960300e+05
1	mean	7.420318e+04
2	std	6.163762e+04
3	min	0.000000e+00
4	25%	4.500000e+04

	Stat	Value
5	50%	6.400000e+04
6	75%	9.000000e+04
7	max	8.706582e+06

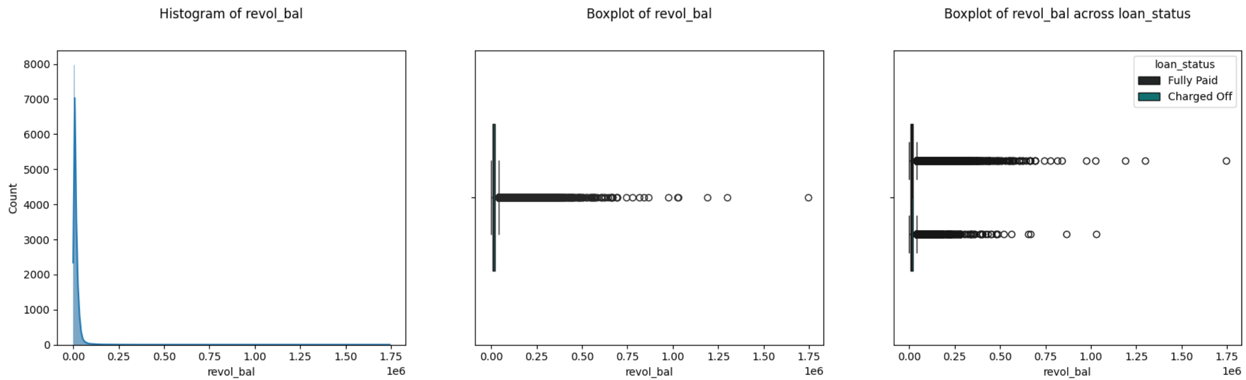


	Stat	Value
0	count	396030.000000
1	mean	17.379514
2	std	18.019092
3	min	0.000000
4	25%	11.280000
5	50%	16.910000
6	75%	22.980000
7	max	9999.000000

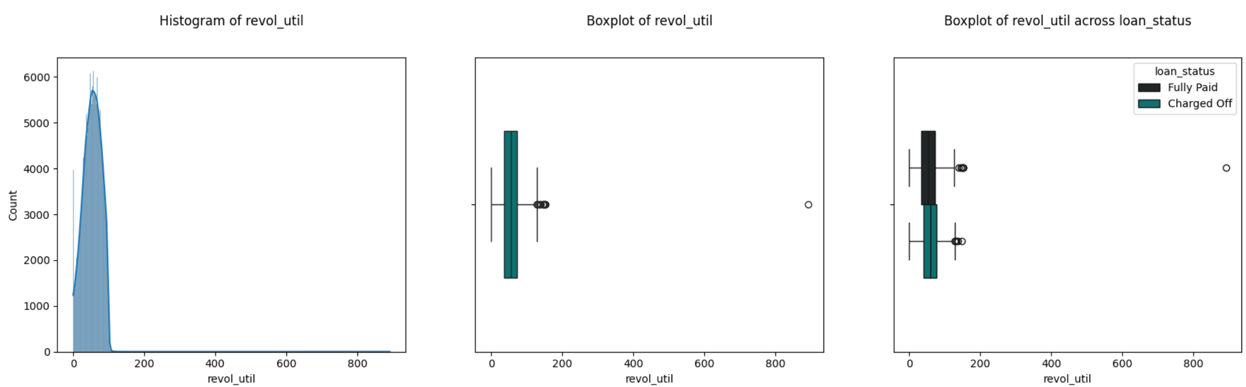


	Stat	Value
0	count	396030.000000
1	mean	11.311153
2	std	5.137649
3	min	0.000000
4	25%	8.000000

	Stat	Value
5	50%	10.000000
6	75%	14.000000
7	max	90.000000

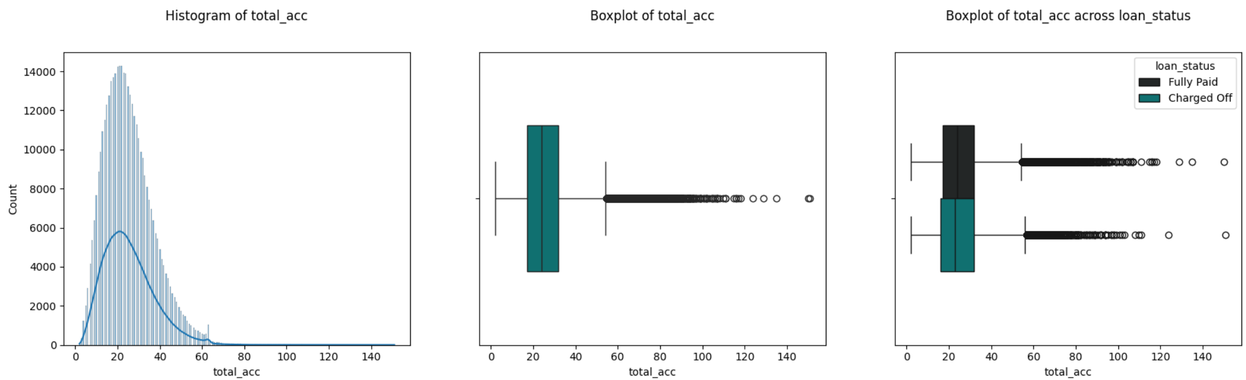


	Stat	Value
0	count	3.960300e+05
1	mean	1.584454e+04
2	std	2.059184e+04
3	min	0.000000e+00
4	25%	6.025000e+03
5	50%	1.118100e+04
6	75%	1.962000e+04
7	max	1.743266e+06



	Stat	Value
0	count	395754.000000
1	mean	53.791749
2	std	24.452193
3	min	0.000000
4	25%	35.800000

	Stat	Value
5	50%	54.800000
6	75%	72.900000
7	max	892.300000



	Stat	Value
0	count	396030.000000
1	mean	25.414744
2	std	11.886991
3	min	2.000000
4	25%	17.000000
5	50%	24.000000
6	75%	32.000000
7	max	151.000000

Key Observations:

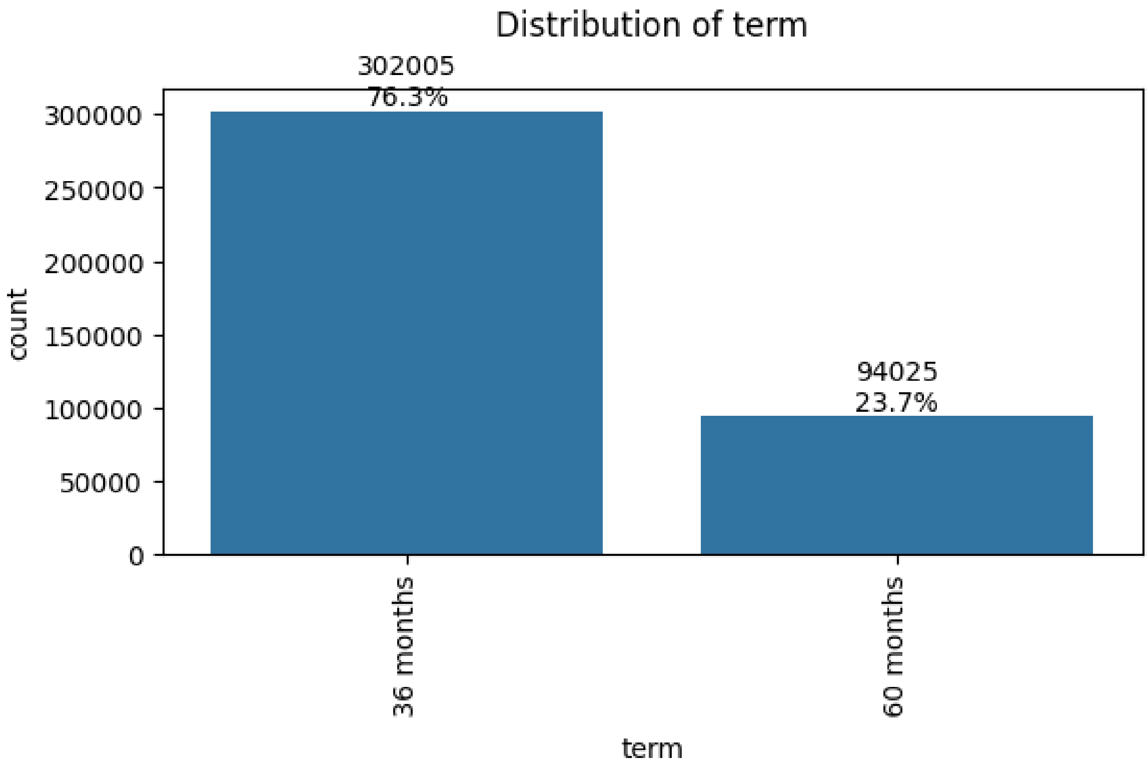
- **Distribution:** Many of the features exhibit right-skewed distributions, indicating a concentration of values towards the lower end and a few instances with very high values (e.g., loan_amnt, int_rate, installment, annual_inc, dti, revol_bal, revol_util, open_acc, total_acc).
- **Outliers:** Outliers are present in most of the features, particularly on the higher end of the value ranges. These outliers could potentially skew the analysis and should be carefully investigated.
- **Loan Status Impact:**
  - int_rate and dti show a strong association with loan status, with "Charged Off" loans generally having higher interest rates and DTI ratios.
  - revol_util also exhibits a strong association with loan status, with higher credit utilization rates being more likely to result in charge-offs.
  - Other features like loan_amnt, installment, annual_inc, open_acc, revol_bal, and total_acc show some relationship with loan status, but the differences between "Charged Off" and "Fully Paid" loans are less pronounced.

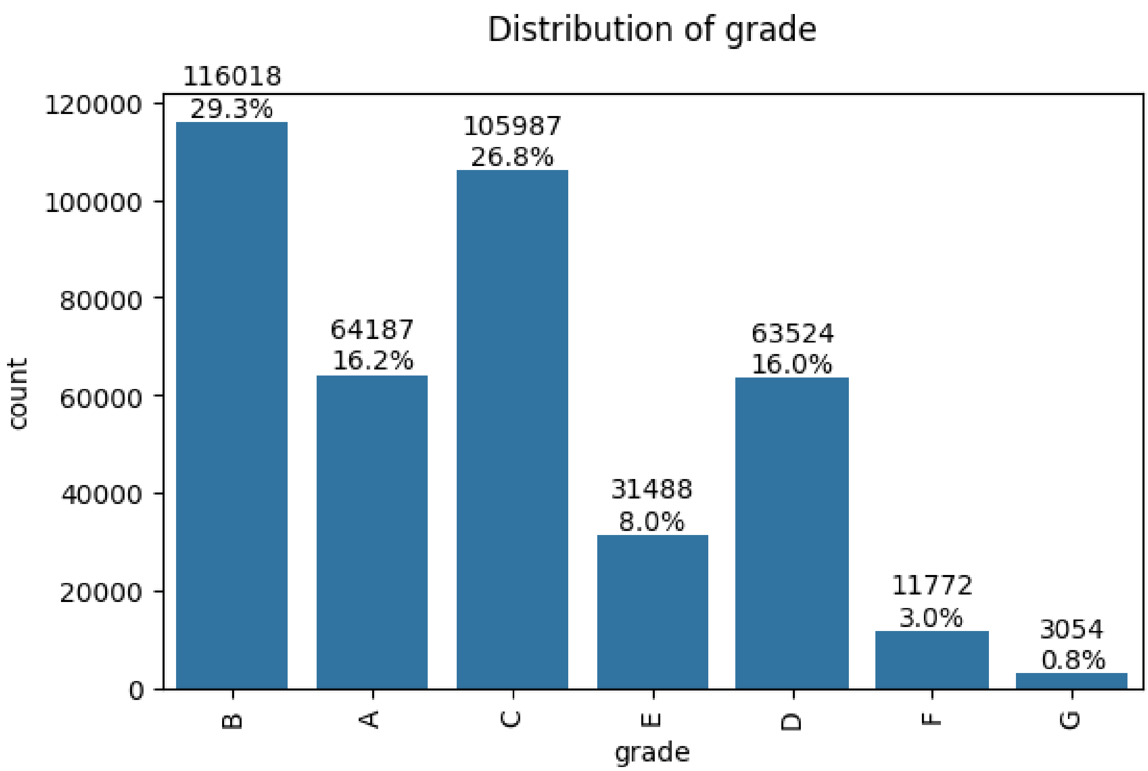
Next Steps:

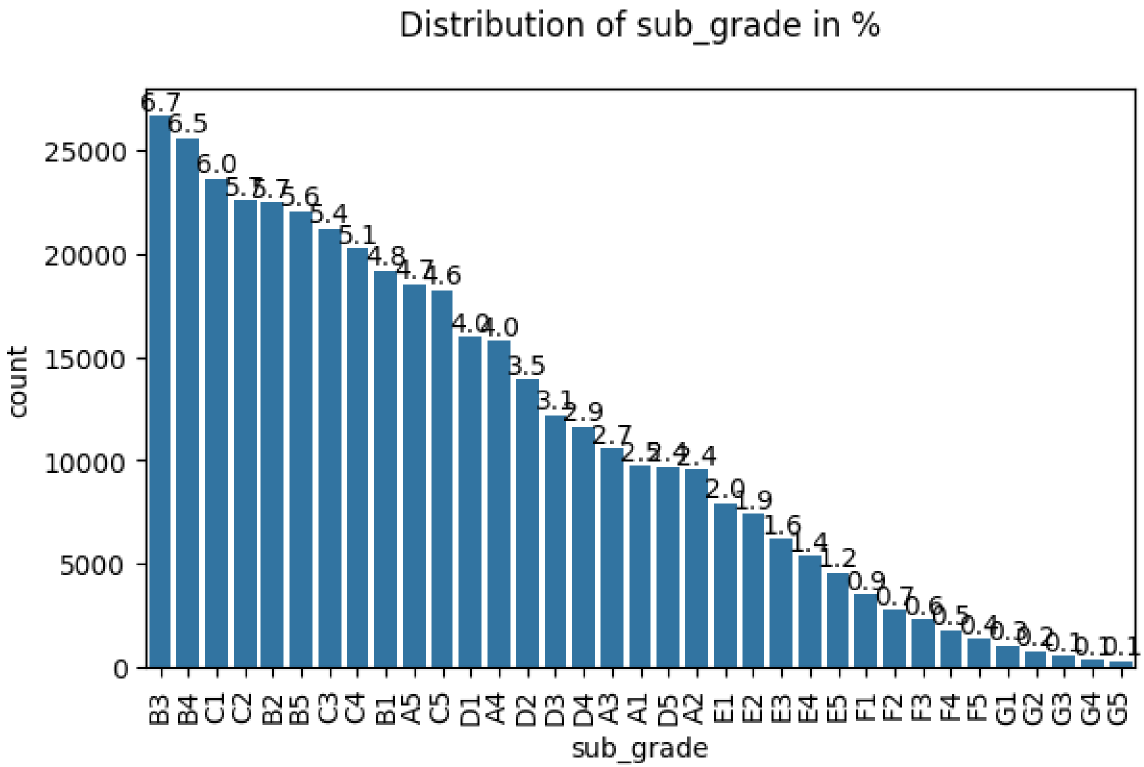
- **Outlier Treatment:**
  - Investigate the causes of outliers in each feature.
  - Consider appropriate outlier treatment strategies, such as:
    - Removal (if justified and after careful analysis)
    - Capping (setting extreme values to a reasonable limit)

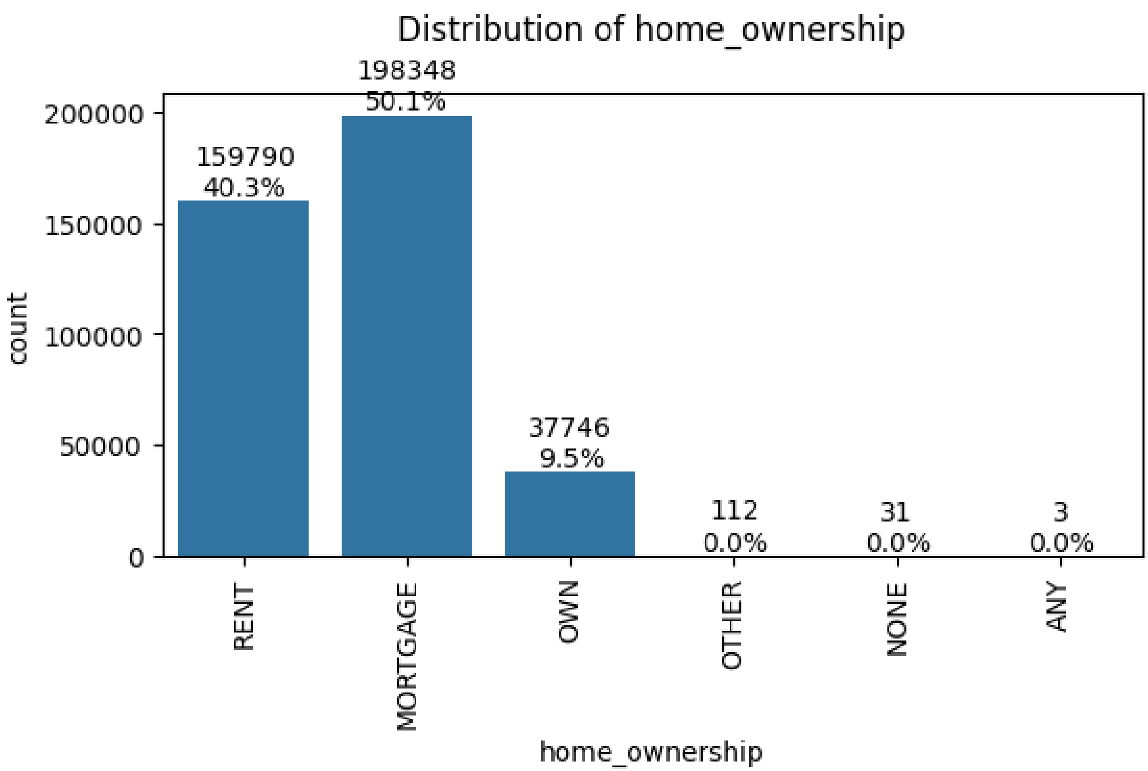
4.2 Analysis of Categorical Columns

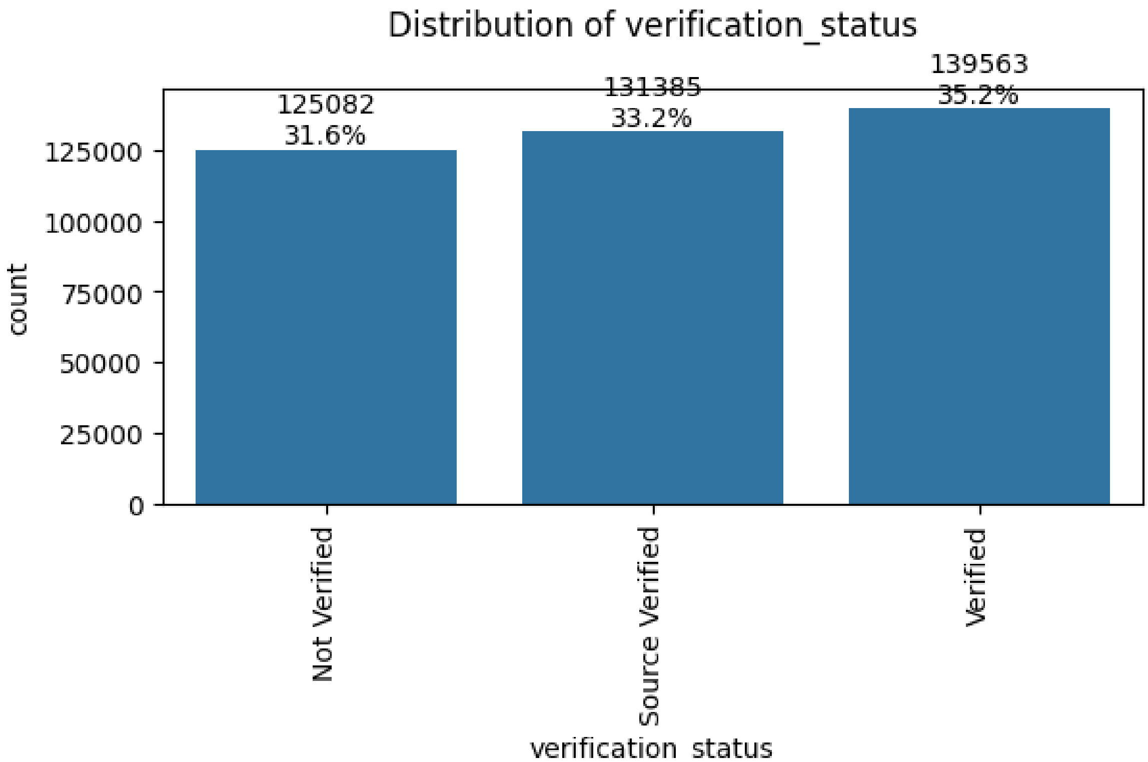
```
categorical = ['term', 'grade','sub_grade','home_ownership',
'verification_status','loan_status','application_type']
for i in categorical:
    if i != 'sub_grade':
      fig, axes = plt.subplots(1,1, figsize=(6, 4), constrained_layout = True)
      plt.title(f'Distribution of {i}', pad= 20, )
      sns.countplot(data=df, x=i)
      for i in axes.patches:
        values = i.get_height()
        percentage = 100 * values / len(df)
        axes.annotate(f'{values:.0f}\n{percentage:.1f}%', (i.get_x() +
i.get_width()/2, i.get_height() + 5), ha = 'center', va = 'bottom')
      plt.xticks(rotation = 90)
    else:
      fig, axes = plt.subplots(1,1, figsize=(6, 4), constrained_layout = True)
      plt.title(f'Distribution of {i} in %', pad= 20)
      sns.countplot(data=df, x=i, order = df[i].value_counts().index)
      for i in axes.patches:
        values = i.get_height()
        percentage = 100 * values / len(df)
        axes.annotate(f'{percentage:.1f}', (i.get_x() + i.get_width()/2,
i.get_height() + 5), ha = 'center', va = 'bottom', fontsize=10)
      plt.xticks(rotation = 90)
    plt.show()
```

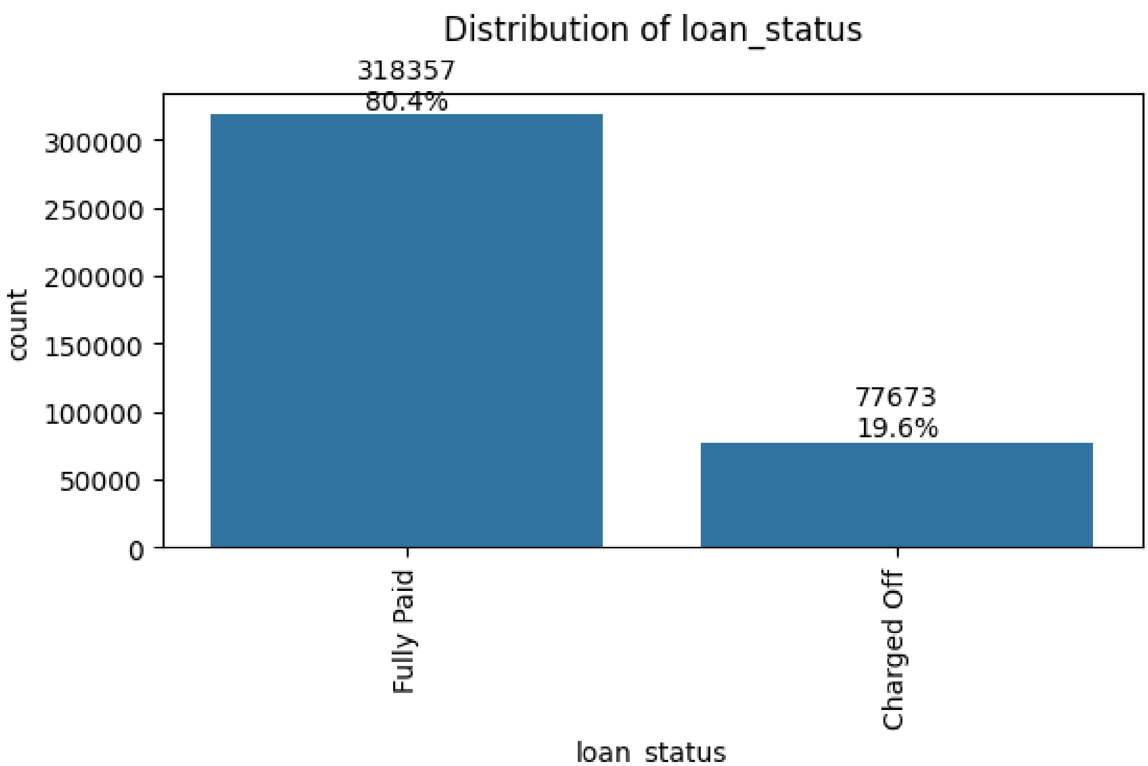

Distribution of term
302005
76.3%
94025
23.7%
300000
250000
200000
150000
100000
50000
0
count
36 months
60 months
term

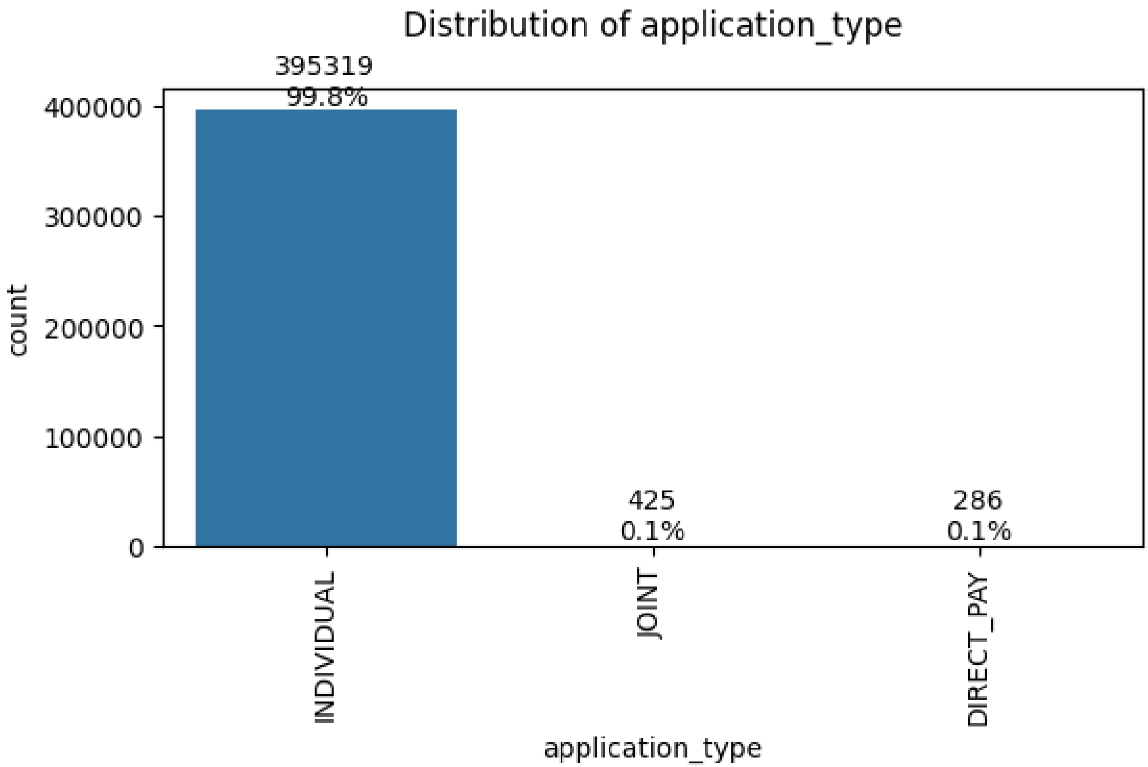

Distribution of grade
116018
29.3%
64187
16.2%
105987
26.8%
31488
8.0%
63524
16.0%
11772
3.0%
3054
0.8%
120000
100000
80000
60000
40000
20000
0
count
B
A
C
E
D
F
G
grade


Distribution of sub_grade in %
count
sub_grade
B3 B4 C1 C2 B2 B5 C3 C4 B1 A5 C5 D1 A4 D2 D3 D4 A3 A1 D5 A2 E1 E2 E3 E4 E5 F1 F2 F3 F4 F5 G1 G2 G3 G4 G5
6.7 6.5 6.0 5.7 5.7 5.6 5.4 5.1 4.8 4.7 4.6 4.0 4.0 3.5 3.1 2.9 2.7 2.5 2.4 2.4 2.0 1.9 1.6 1.4 1.2 0.9 0.7 0.6 0.5 0.4 0.3 0.2 0.1 0.1 0.1


Distribution of home_ownership
count
home_ownership
RENT
159790
40.3%
MORTGAGE
198348
50.1%
OWN
37746
9.5%
OTHER
112
0.0%
NONE
31
0.0%
ANY
3
0.0%


Distribution of verification_status
125082
31.6%
131385
33.2%
139563
35.2%
count
0
25000
50000
75000
100000
125000
Not Verified
Source Verified
Verified
verification_status


Distribution of loan_status
318357
80.4%
77673
19.6%
count
0
50000
100000
150000
200000
250000
300000
Fully Paid
Charged Off
loan_status



Key Observations:

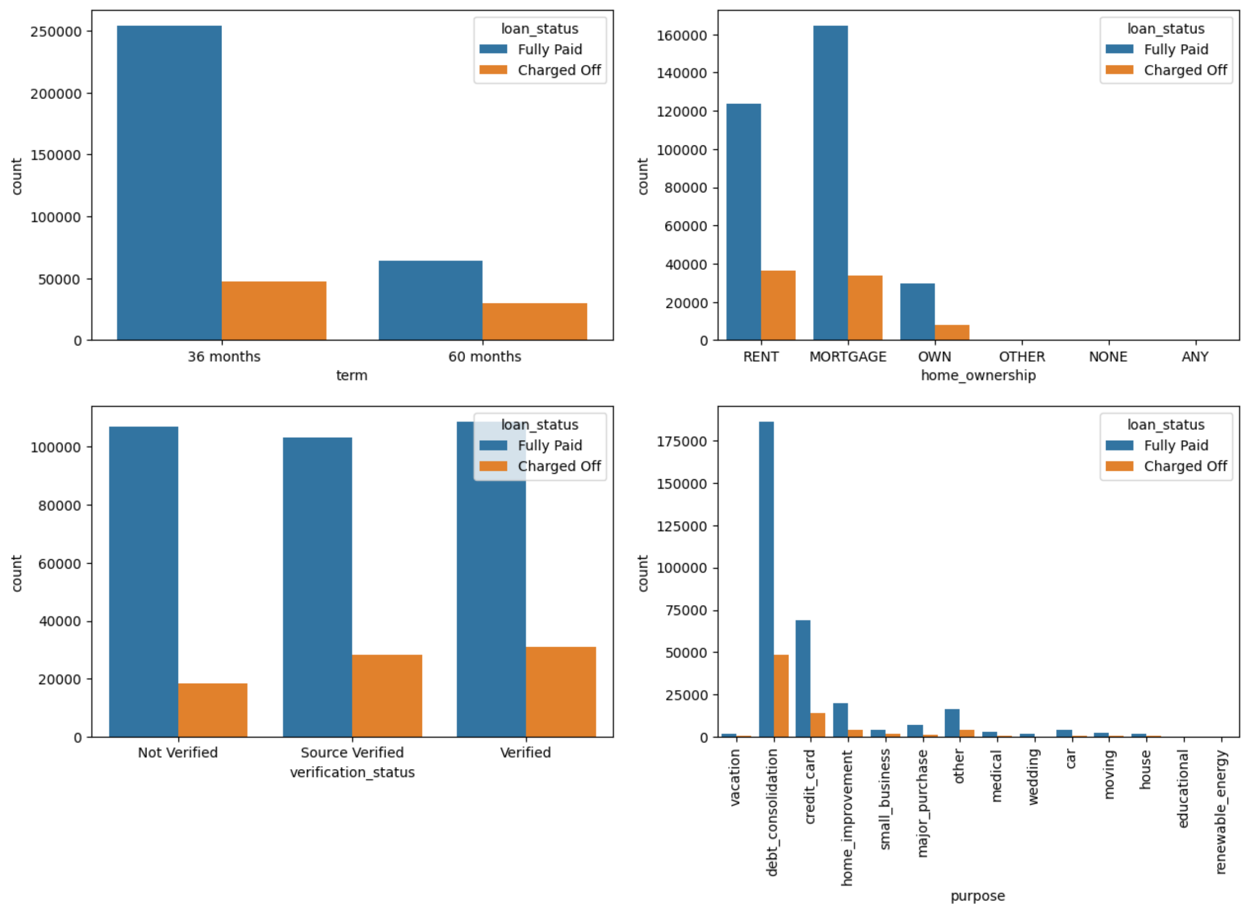
- **Loan Term Preference:** A 36-month loan term is significantly preferred over a 60-month term, accounting for approximately 76% of loans.
- **Grade Distribution:** Borrowers are predominantly concentrated in grades C and B, while grade G represents a very small proportion (0.8%).
- **Subgrade Distribution:** The subgrade distribution follows a similar pattern to the grade distribution, with each grade further subdivided into five groups, exhibiting a declining frequency.
- **Home Ownership:** "Mortgage" is the most common home ownership status, accounting for 50% of borrowers. "Rent" is the second most prevalent status.
- **Verification Status:** Approximately one-third of applicants have not verified their income.
- **Loan Performance:** 80.4% of loans have been "Fully Paid," while 19.6% have been "Charged Off."
- **Application Type:** The vast majority (99.8%) of applications are submitted by individuals.

Next Steps: - **Investigate the Impact of Non-Verified Income:** - Analyze the relationship between non-verified income status and loan performance across different customer segments. - Determine if non-verified income applicants have a higher likelihood of loan default.

4.3 Important features across the Target variable

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

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

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

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

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

plt.show()
```

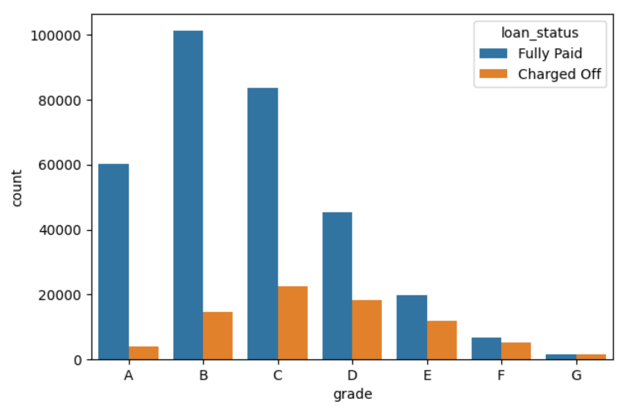


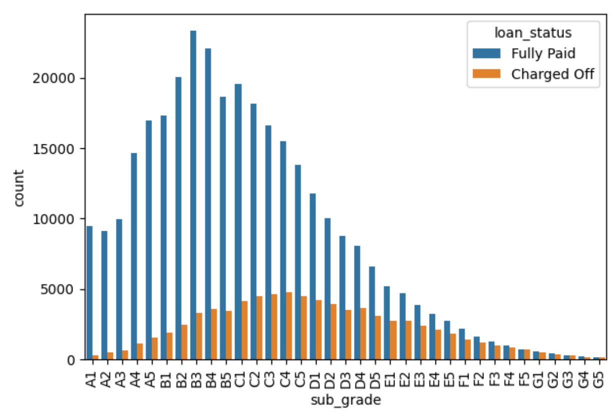
Key Observations:

- **Term:** A clear preference for shorter loan terms is evident, with 36-month terms being significantly more common than 60-month terms. This preference also seems to be associated with lower default rates.

- **Home Ownership:** Borrowers with mortgages have the highest number of loans, followed by renters. Homeowners appear to have slightly lower default rates compared to renters.
- **Verification Status:** A substantial portion of borrowers have not verified their income. While this category has a higher number of loans, it also shows a slightly less proportion of defaults compared to "Source Verified" and "Verified" income.
- **Purpose:** The primary purpose for loans is "debt consolidation," followed by "credit card" and "home improvement." Loans taken for "debt consolidation" and "small business" appear to have higher default rates.

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

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

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

plt.show()
```



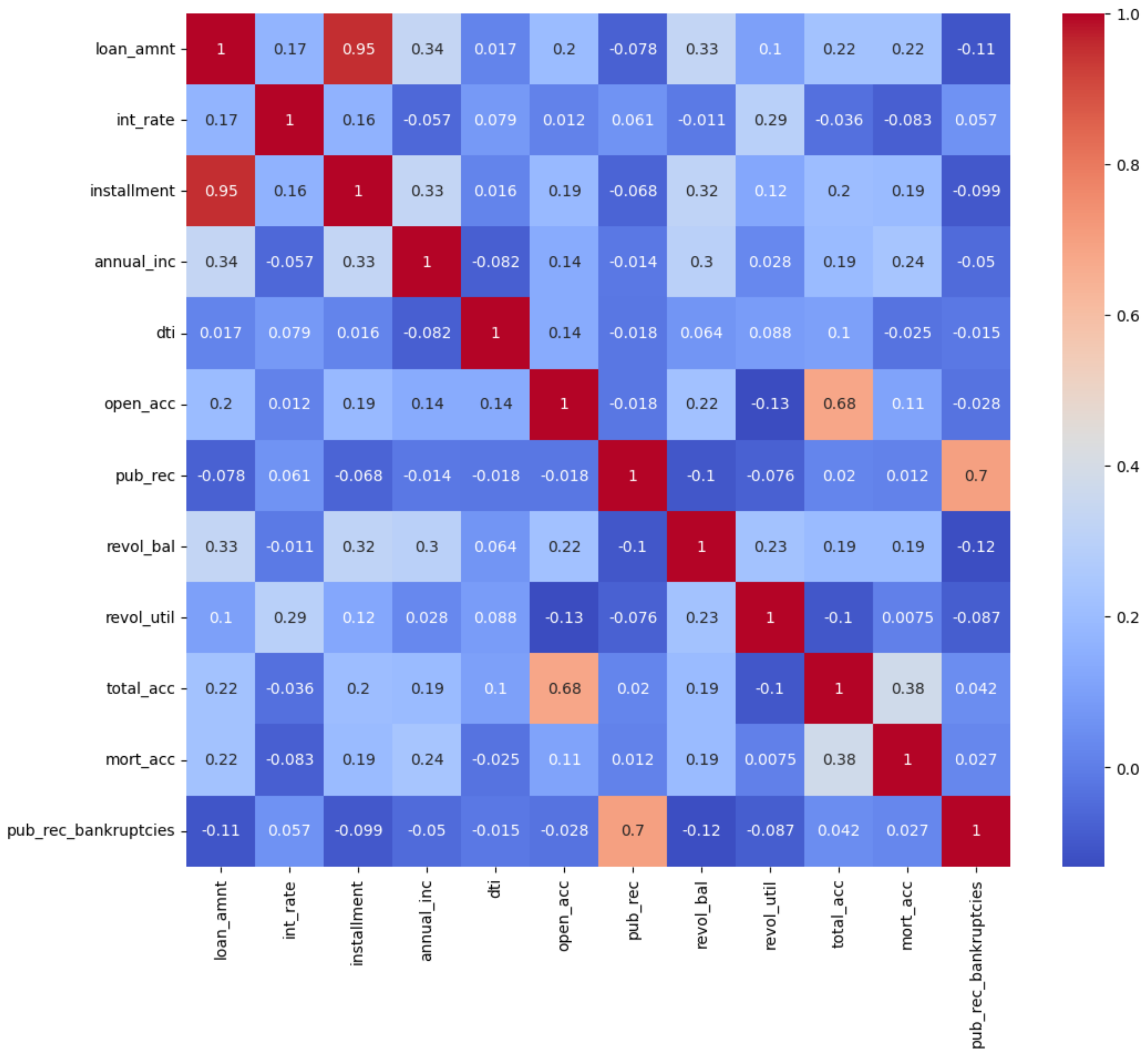


Key Observations:

- **Grade Distribution & Default Rates:**
  - Grade C has the highest number of loans, followed by grade B.
  - Grades A and G have the lowest number of loans.
  - Default rates appear to increase with decreasing grade (from A to G). Grades G and F have the highest default rates.
- **Sub-grade Distribution & Default Rates:**

- The distribution of loans across sub-grades follows a similar pattern to the grade distribution.
- Within each grade, the default rate generally increases as the sub-grade letter moves further down the alphabet (e.g., A1 to A5).
- Sub-grades G5 and F5 have the highest default rates.

```
plt.figure(figsize = (12,10))
sns.heatmap(df.select_dtypes("number").corr(), annot =True, cmap = 'coolwarm')
```



5.0 Data Preprocessing

Duplicate Value check

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

0

5.1 Handling the Missing values

###5.1.1 Making an informed decision on handling the missing values

Can we drop the records with null values

```
missing_value_perc()
```

	Missing Values Percentage
emp_title	5.79
emp_length	4.62
title	0.44
revol_util	0.07
mort_acc	9.54
pub_rec_bankruptcies	0.14

Key Observations - The "mort_acc" column has the highest percentage of missing values compared to other columns in the dataset.

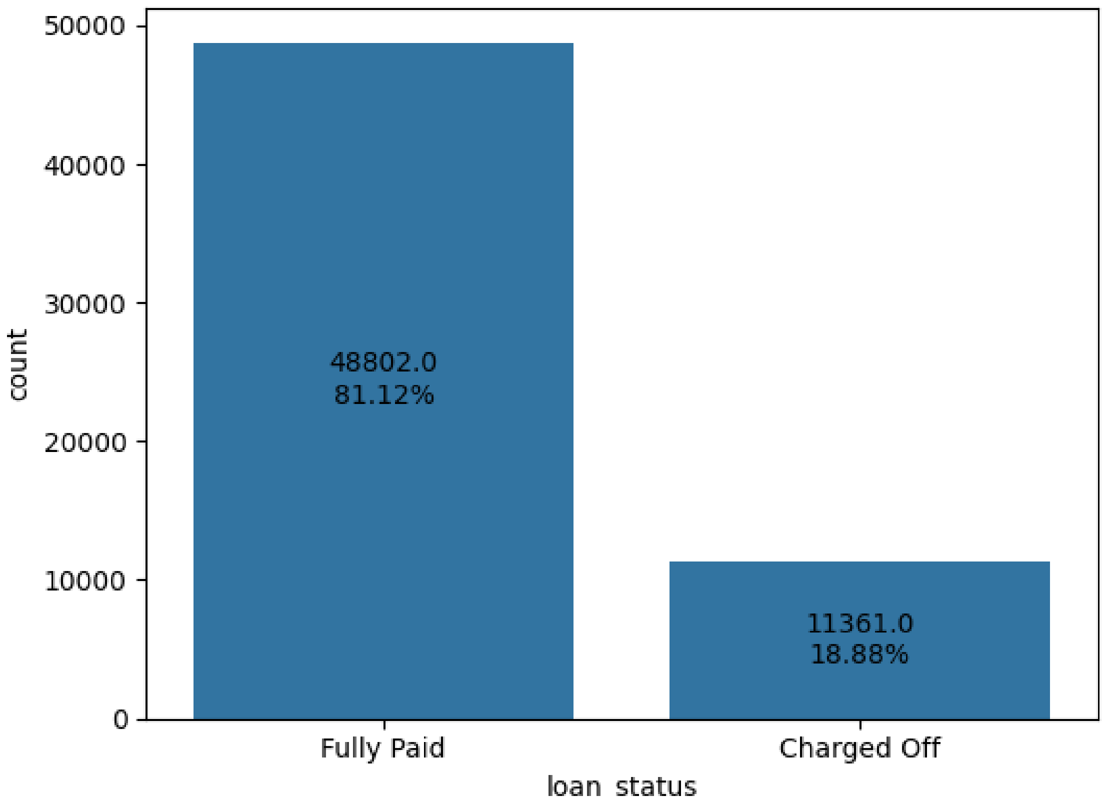
Next Steps - Calculate Impact of Dropping "mort_acc" Rows: - Determine the percentage of data that would be lost if rows with missing values in the "mort_acc" column are dropped. - Assess the potential impact of data loss on the analysis and model building.

5.1.2 Null records percentage across the dataset

```
# Function to calculate the missing records percentage across the dataset
def missing_records(target):
  missing_records = df[df.isnull().any(axis=1)]
  plt = sns.countplot(data = missing_records, x = missing_records[target])
  a = np.round(100 * len(missing_records)/len(df),2)
  print(f'Records with null values if dropped will constitute to {a}% of the
  overall dataset')
  if a > 5:
    print('Dropping the records is not the best strategy yet')
  else:
    print('We can go ahead and drop the records as we have enough records')
  for i in plt.patches:
    values = i.get_height()
    percentage = 100 * values/len(missing_records)
```

```
    plt.annotate(f'{values}\n{percentage:.2f}%', (i.get_x() + i.get_width()/2,
i.get_height()/2), ha = 'center', va = 'center', fontsize = 10)


missing_records('loan_status')
```

```
Records with null values if dropped will constitute to 15.19% of the overall
dataset
Dropping the records is not the best strategy yet
```



Key Observations - The distribution of missing records closely mirrors the overall distribution of the loan_status variable, suggesting that imputation using central tendencies could be a viable strategy. - Dropping the missing records is not feasible due to the significant percentage (15.19%) they constitute. Instead, a hybrid approach of selective imputation can be explored to reduce the overall percentage of missing data.

Next Steps 1. Prioritize Columns with High Missing Percentages - Focus on handling the top contributors to missing data, starting with emp_title, title, and mort_acc. 2. Imputation Strategies - Categorical Variables (emp_title, title): Impute missing values with "Unknown" instead of the mode since these variables will not be included in model training. - Numerical Variable (mort_acc): Impute missing values using central tendencies (mean or median), as it constitutes a significant 9.54% of the data. - Ordinal Variable (emp_length): Replace missing values with 0, as this variable will be excluded from the final model. 3. Iterative Approach - After addressing the key contributors to missing data, reassess the overall missing record percentage. If the percentage drops below a reasonable threshold, consider removing any residual records with missing values.

###**5.1.3 Data Imputation**

```
df.loc[df['emp_title'].isnull(),'emp_title'] = 'Unknown'
df.loc[df['title'].isnull(),'title'] = 'Unknown'
df.loc[df['emp_length'].isnull(),'emp_length'] = 0
missing_value_perc()
```

	Missing Values Percentage
revol_util	0.07
mort_acc	9.54
pub_rec_bankruptcies	0.14

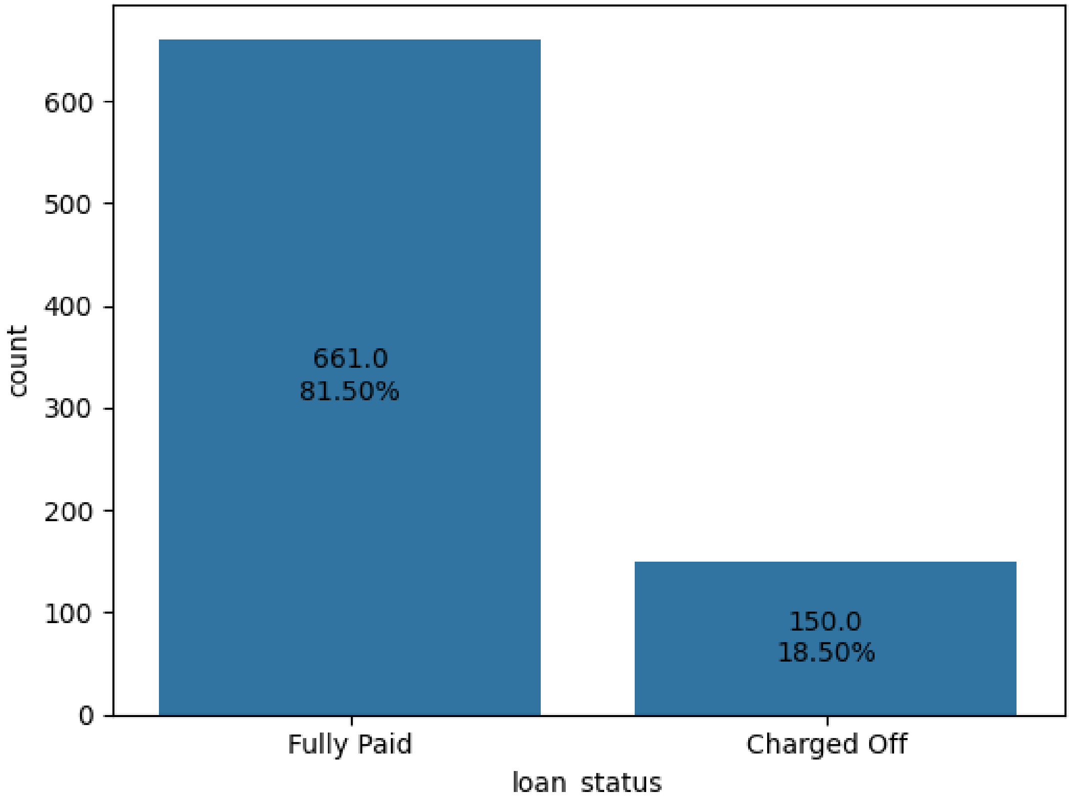
```
total_acc_avg=df.groupby(by='total_acc').mort_acc.mean()
# saving mean of mort_acc according to total_acc_avg
def fill_mort_acc(total_acc,mort_acc):
    if np.isnan(mort_acc):
        return total_acc_avg[total_acc].round()
    else:
        return mort_acc
df['mort_acc']=df.apply(lambda x:
fill_mort_acc(x['total_acc'],x['mort_acc']),axis=1)
missing_value_perc()
```

	Missing Values Percentage
revol_util	0.07
pub_rec_bankruptcies	0.14

##5.2 Reassesing the Missing records percentage after handling major contributors.

```
missing_records('loan_status')
```

```
Records with null values if dropped will constitute to 0.2% of the overall
dataset
We can go ahead and drop the records as we have enough records
```




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

Missing Values Percentage

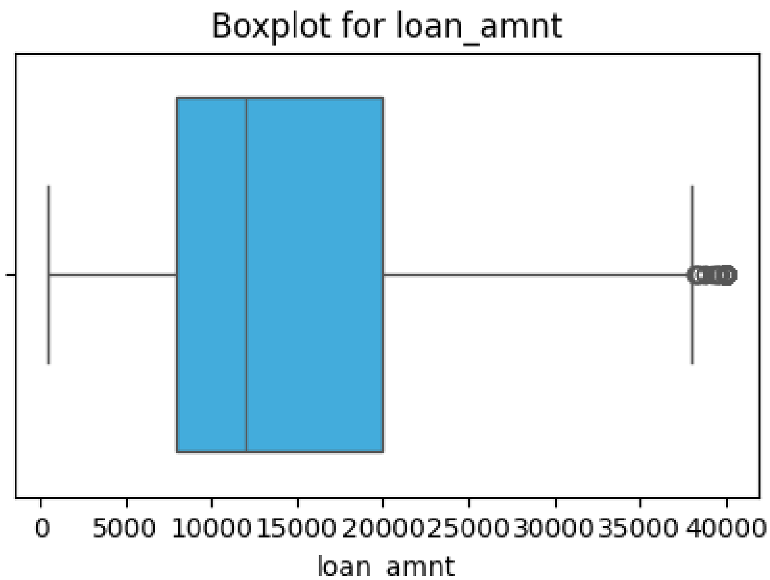
##5.3 Handling the Outlier values

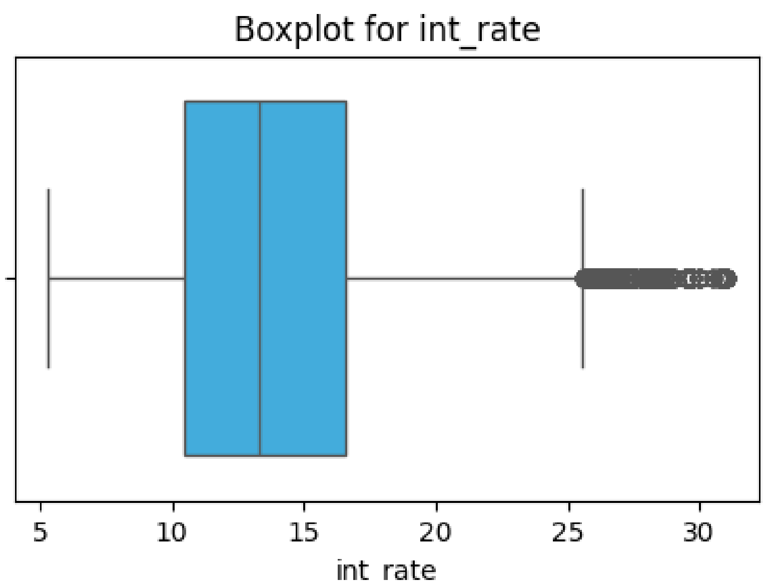
5.3.1 Outlier Detection

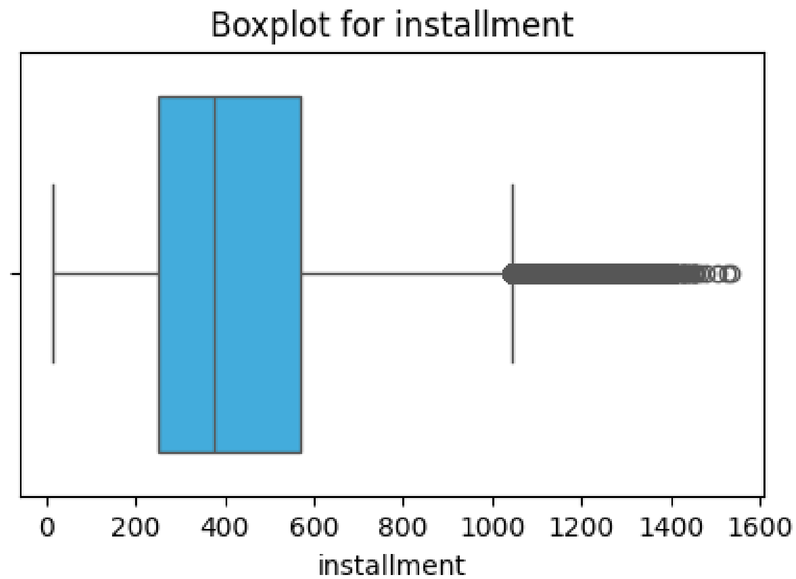
```
numerical = df.select_dtypes(include = 'number').columns
categorical = ['term', 'grade', 'sub_grade', 'home_ownership',
'verification_status', 'loan_status', 'purpose', 'application_type']
```

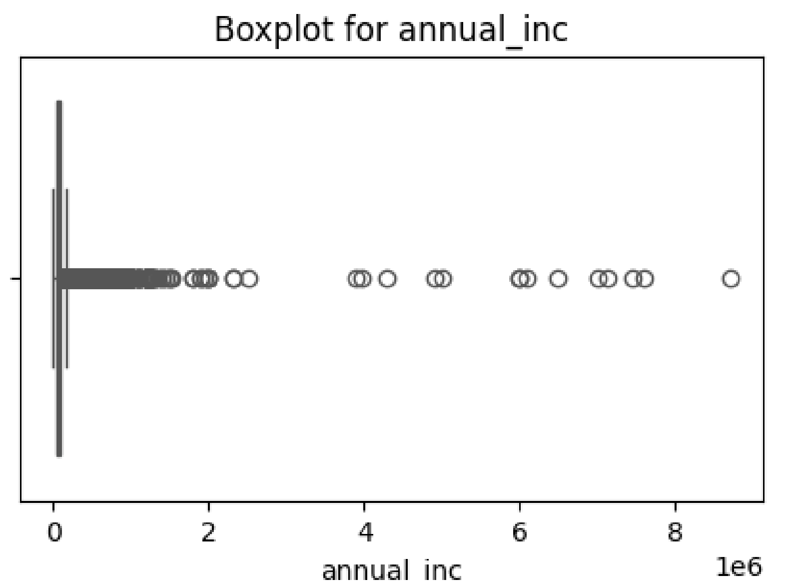
```
def box_plot(col):
    if col in df.columns:
        plt.figure(figsize=(5, 3))
        sns.boxplot(x=df[col],color="#29B6F6")
        plt.title('Boxplot for {}'.format(col))
        plt.show()
```

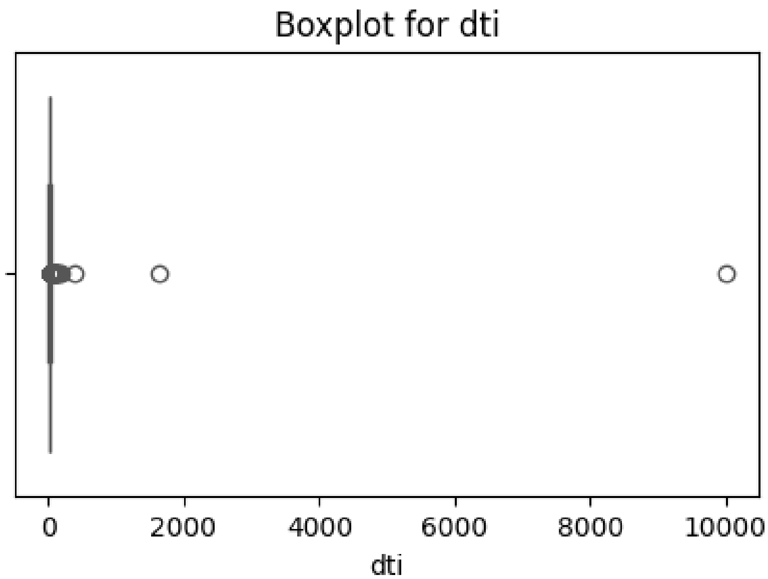
```
    else:
        print(f"Column '{col}' not found in the DataFrame.")

for col in numerical:
    box_plot(col)
```

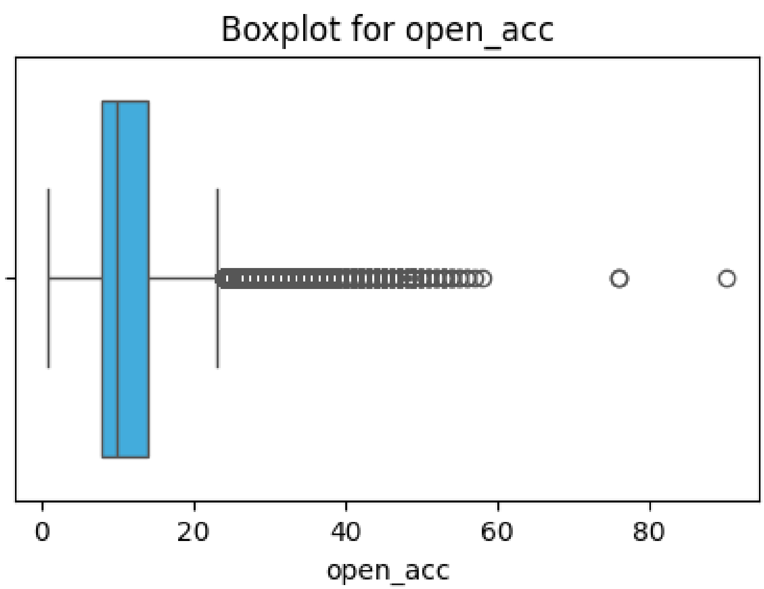


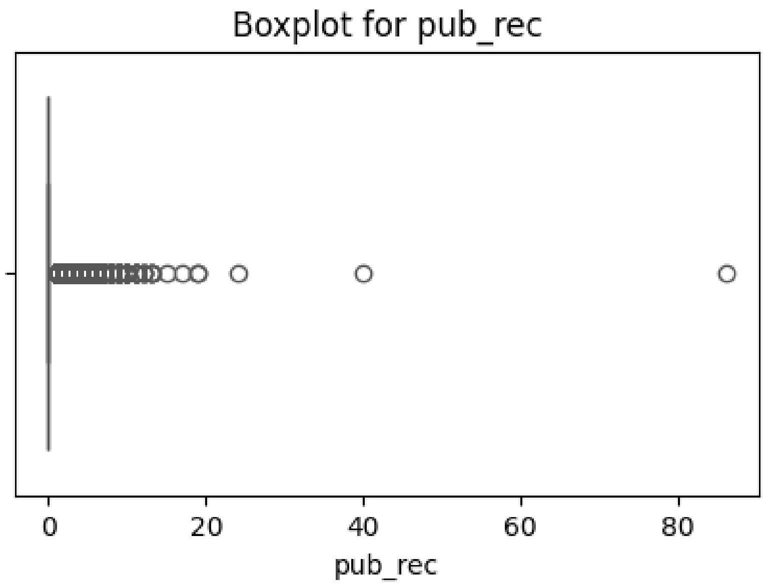


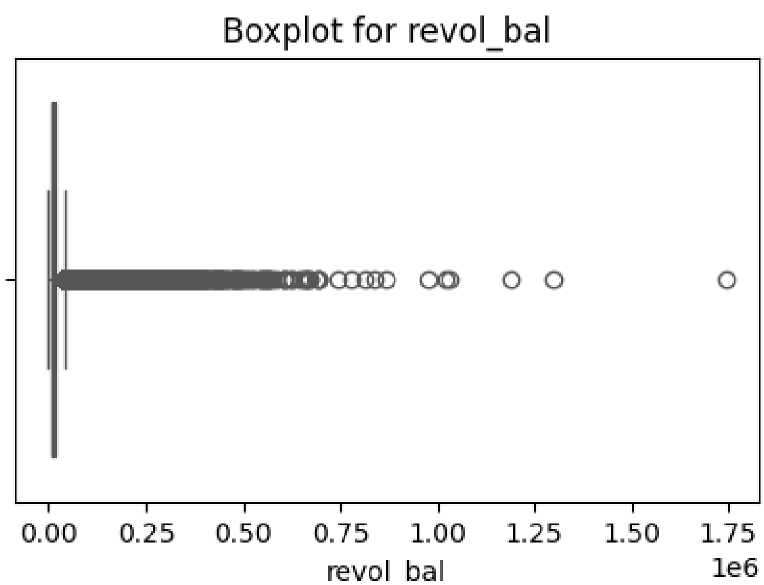

Boxplot for installment
0
200
400
600
800
1000
1200
1400
1600
installment

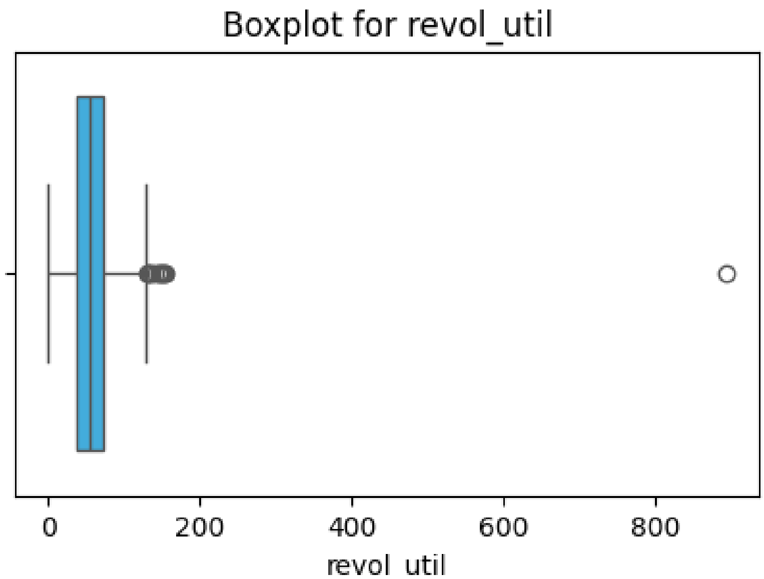

Boxplot for annual_inc
0
2
4
6
8
1e6
annual_inc

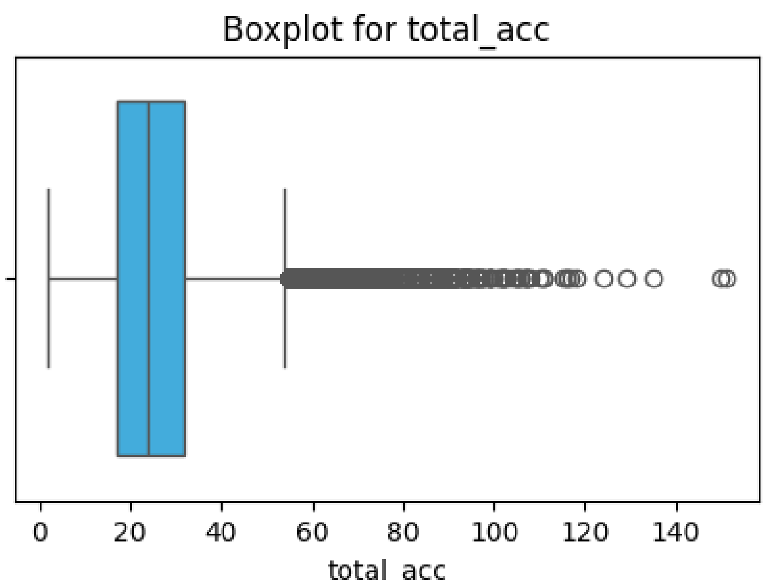

Boxplot for dti
0
2000
4000
6000
8000
10000
dti

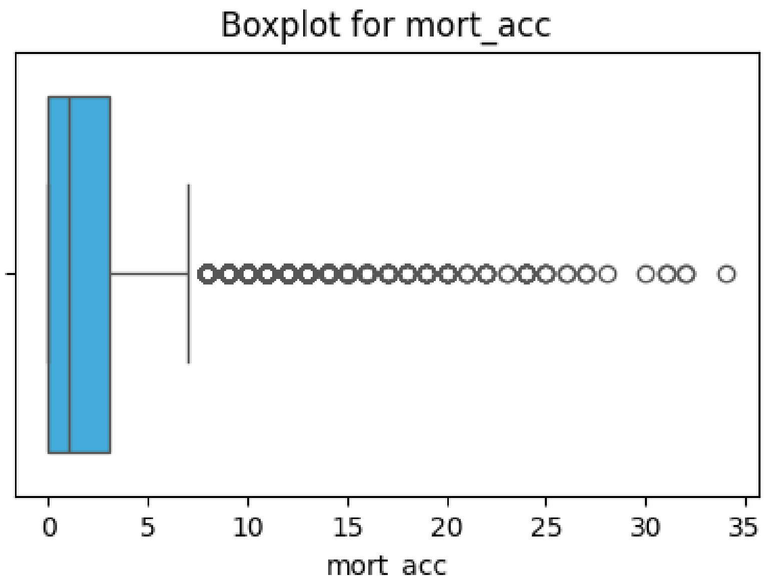

Boxplot for open_acc
0
20
40
60
80
open_acc

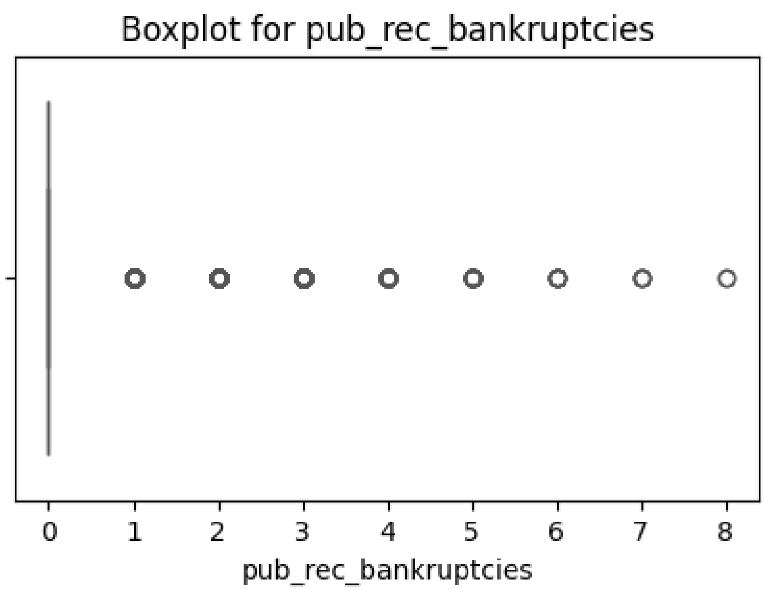

Boxplot for pub_rec
0
20
40
60
80
pub_rec


Boxplot for revol_bal
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
1e6
revol_bal


Boxplot for revol_util
0
200
400
600
800
revol_util


Boxplot for total_acc
0
20
40
60
80
100
120
140
total_acc



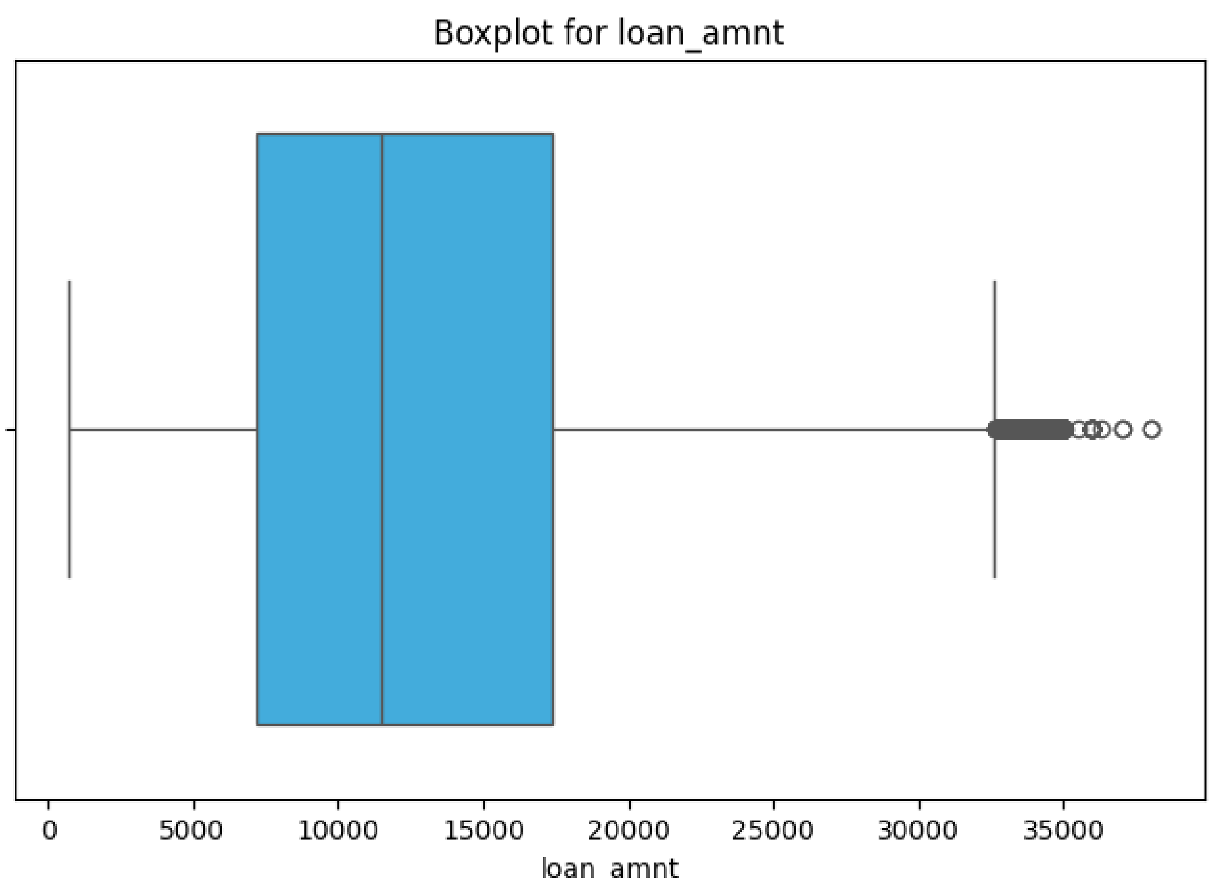


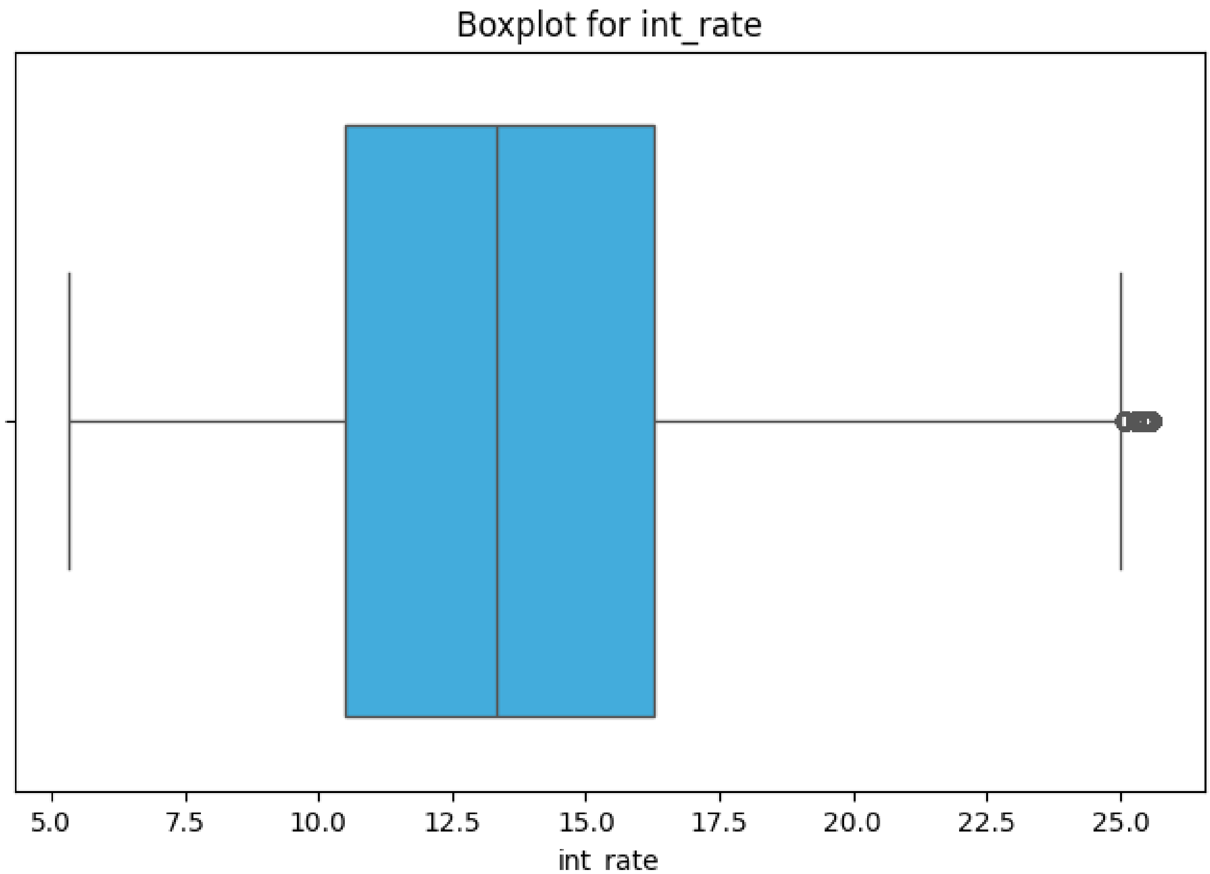

```
df.select_dtypes(include = 'number').columns
```

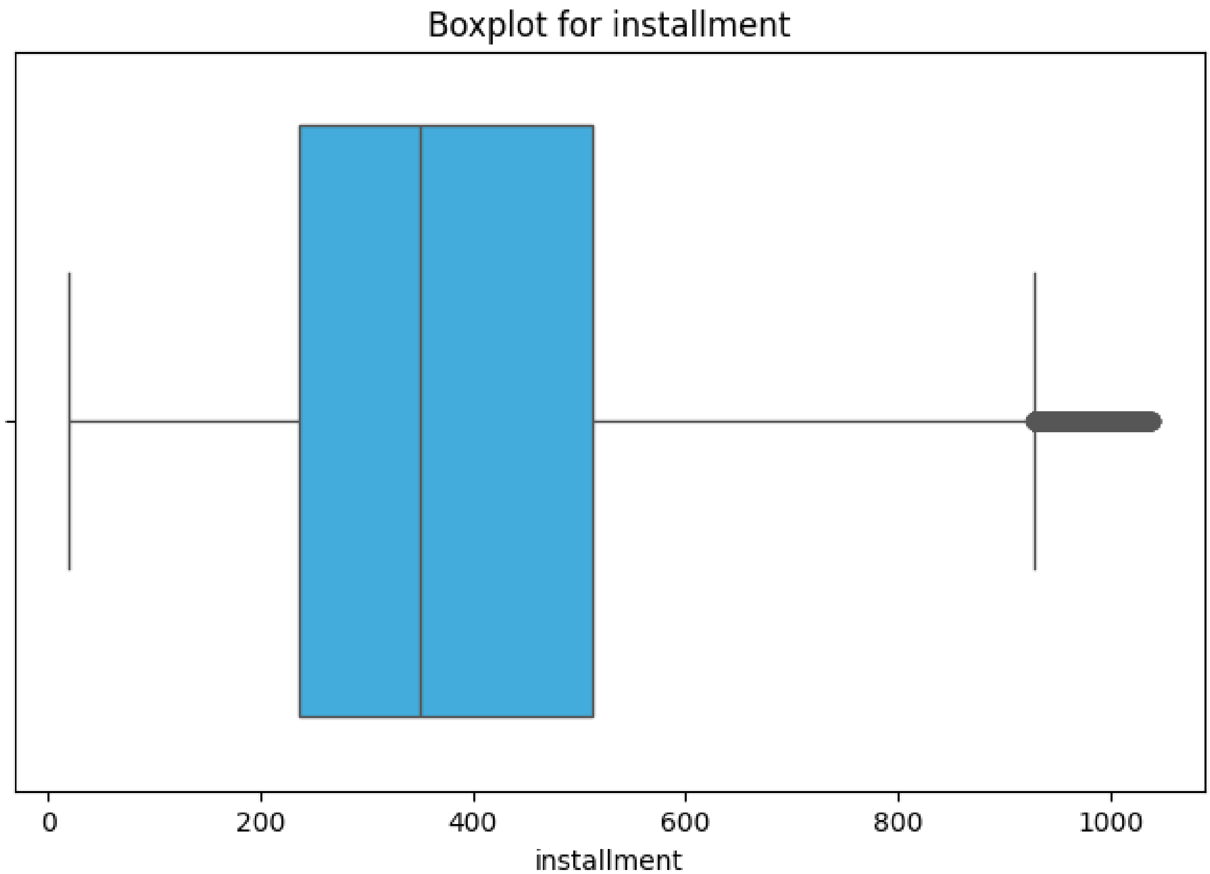
```
Index(['loan_amnt', 'int_rate', 'installment', 'annual_inc', 'dti', 'open_acc',
       'pub_rec', 'revol_bal', 'revol_util', 'total_acc', 'mort_acc',
       'pub_rec_bankruptcies'],
      dtype='object')
```

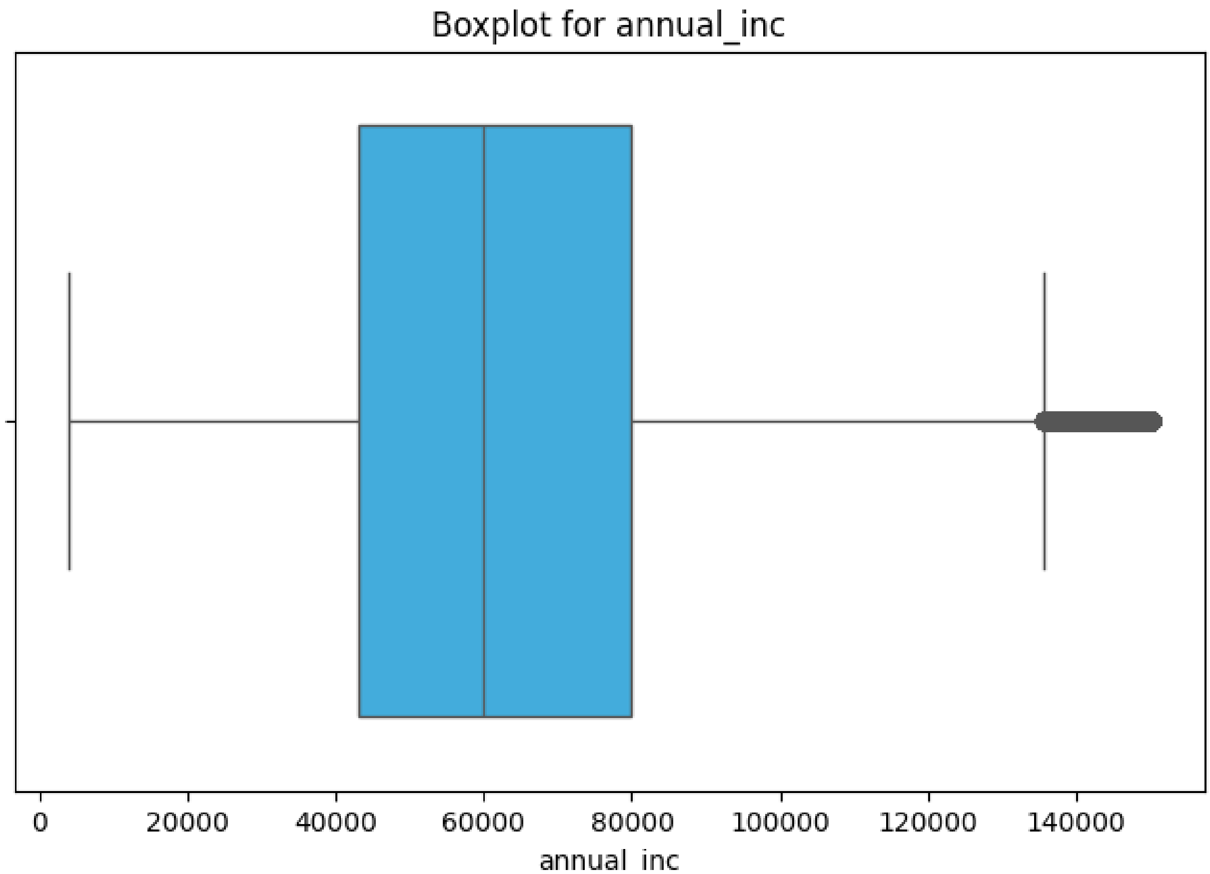
```
# Outlier treatment
new_num_cols=['loan_amnt', 'int_rate', 'installment', 'annual_inc', 'dti',
'open_acc', 'revol_bal', 'revol_util', 'total_acc']
for col in new_num_cols:
    if col in df.columns:
```

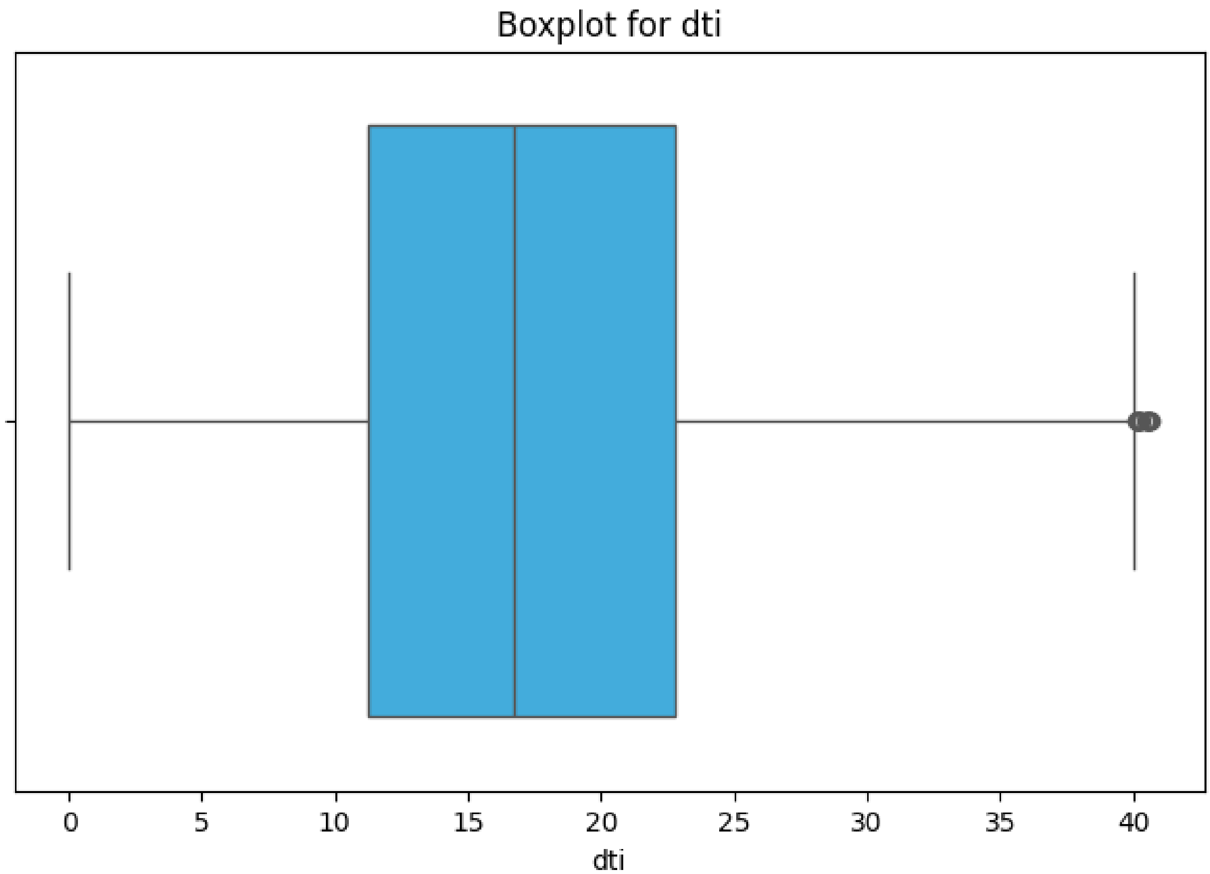
```
        Q1 = df[col].quantile(0.25)
        Q3 = df[col].quantile(0.75)
        IQR = Q3 - Q1
        lower_limit = Q1 - 1.5 * IQR
        upper_limit = Q3 + 1.5 * IQR
        df = df[(df[col] >= lower_limit) & (df[col] <= upper_limit)]

def box_plot(col):
    if col in df.columns:
        plt.figure(figsize=(8, 5))
        sns.boxplot(x=df[col],color="#29B6F6")
        plt.title('Boxplot for {}'.format(col))
        plt.show()
    else:
        print(f"Column '{col}' not found in the DataFrame.")

for col in new_num_cols:
    box_plot(col)
df.shape
```



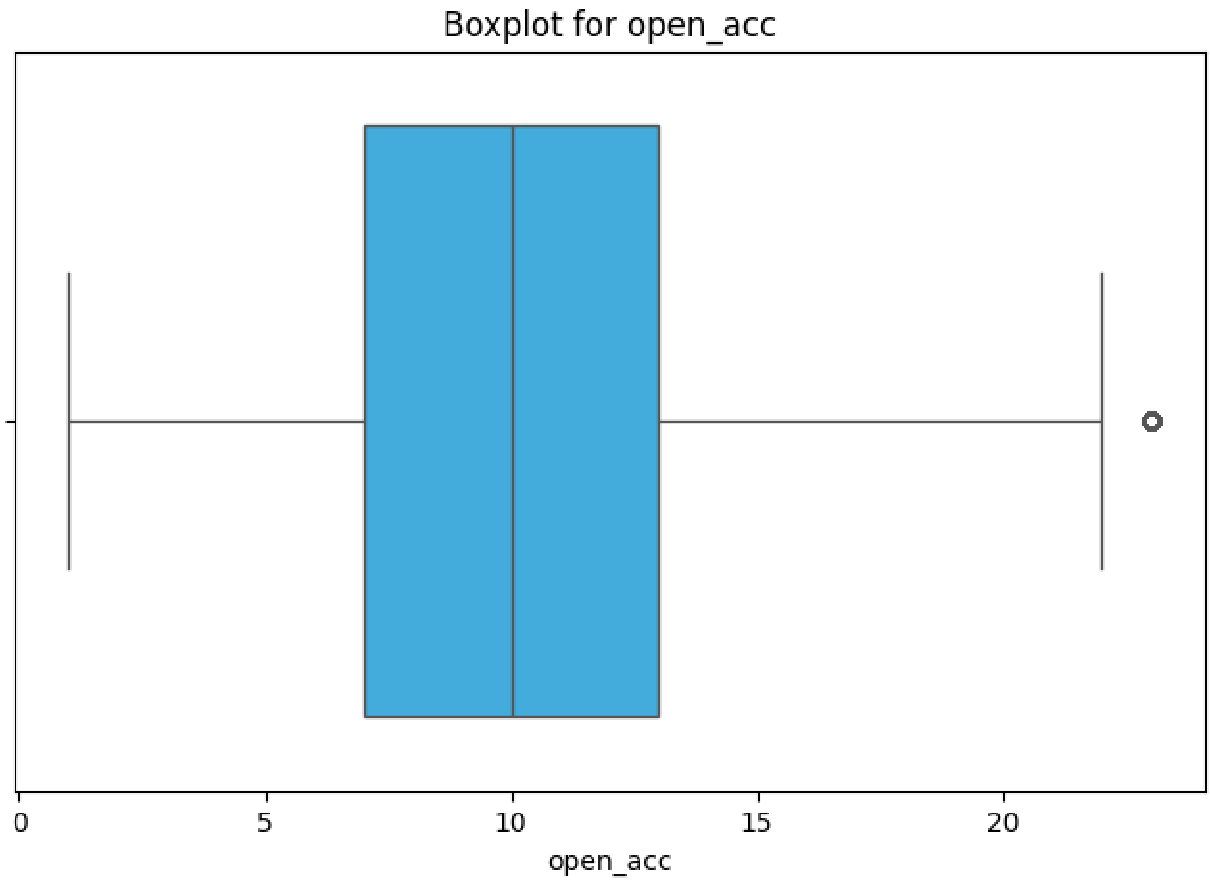

Boxplot for int_rate
5.0
7.5
10.0
12.5
15.0
17.5
20.0
22.5
25.0
int_rate


Boxplot for installment
0
200
400
600
800
1000
installment

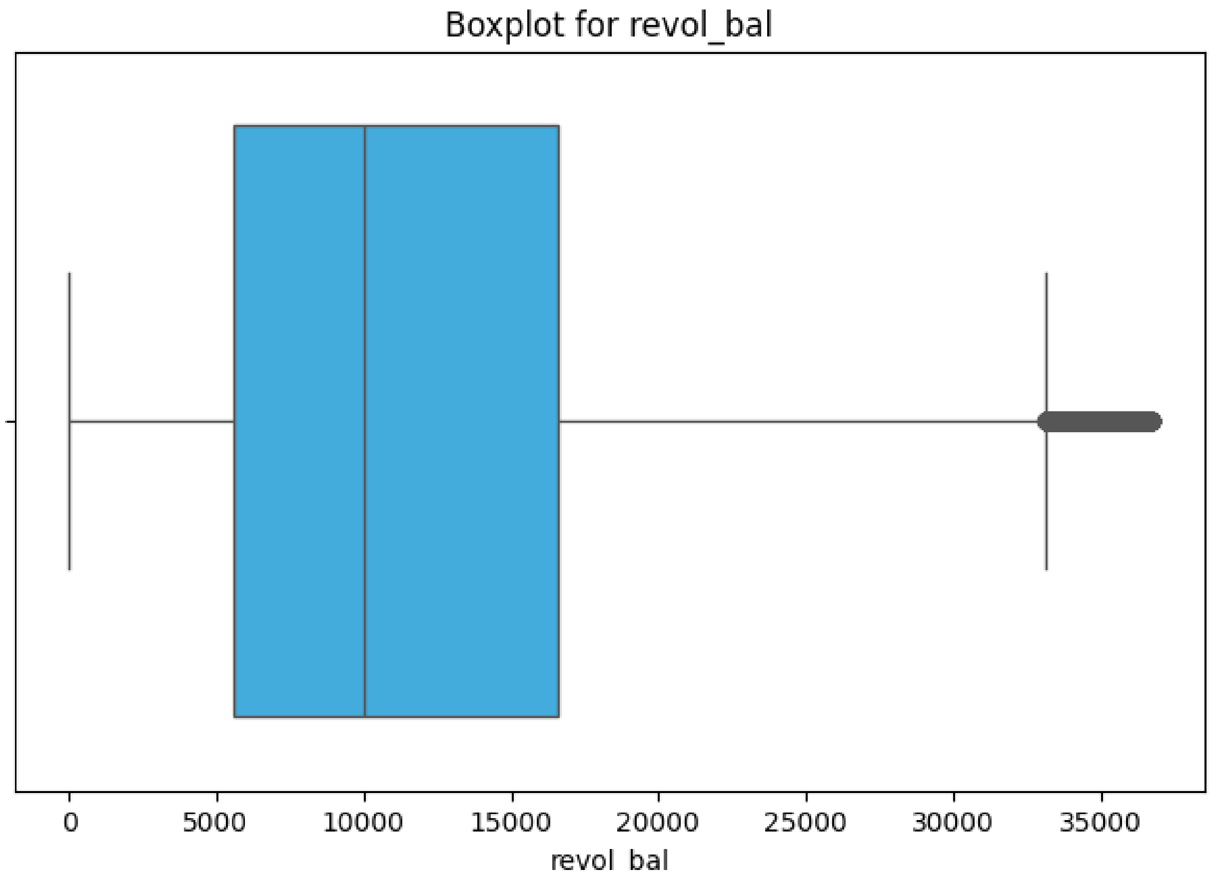

Boxplot for annual_inc
0
20000
40000
60000
80000
100000
120000
140000
annual_inc

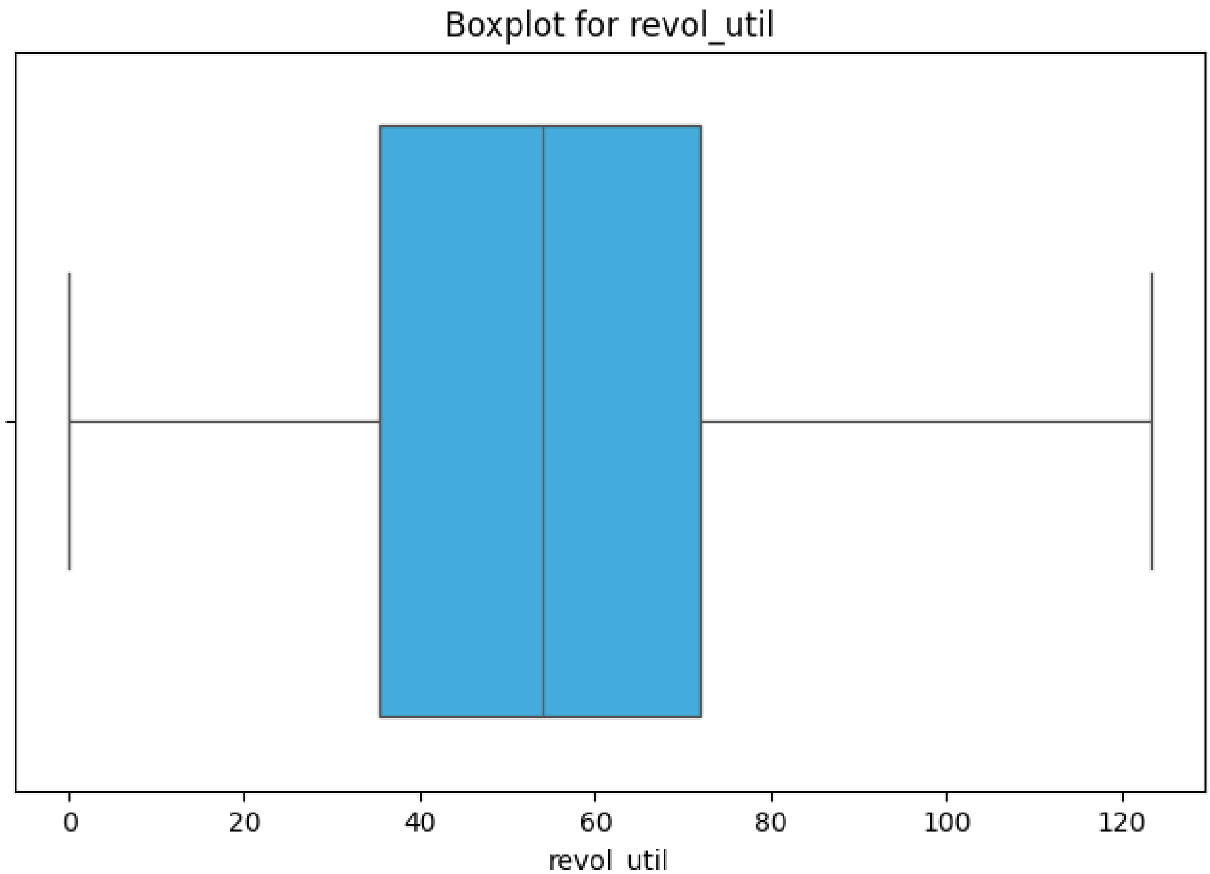


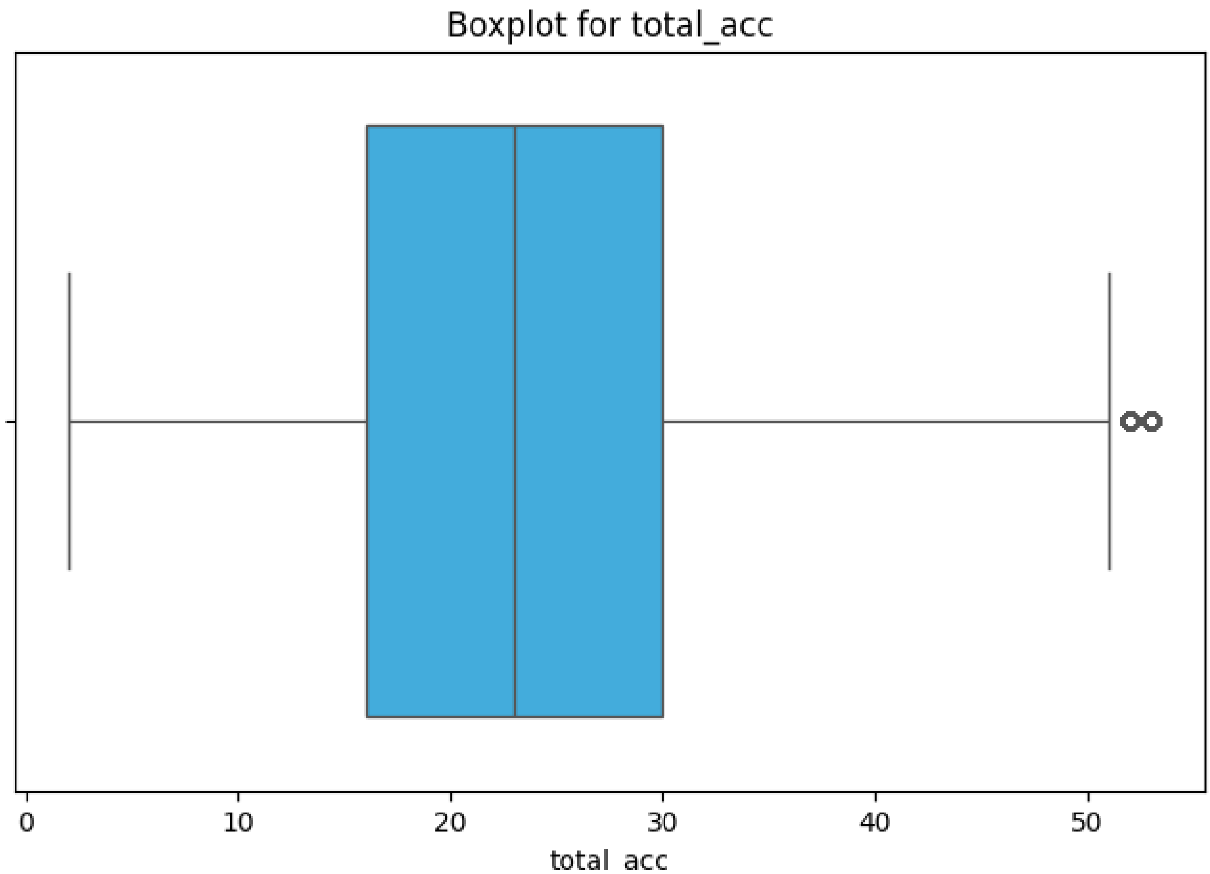
Boxplot for dti
0
5
10
15
20
25
30
35
40
dti



Boxplot for open_acc
0
5
10
15
20
open_acc


Boxplot for revol_bal
0
5000
10000
15000
20000
25000
30000
35000
revol_bal


Boxplot for revol_util
0
20
40
60
80
100
120
revol_util



###5.3.2 Feature Engineering

Imputing values for categorical variables

```
df['pub_rec'] = [1 if i > 1 else 0 for i in df['pub_rec']]
df['mort_acc'] = [1 if i > 1 else 0 for i in df['mort_acc']]
df['pub_rec_bankruptcies'] = [1 if i > 1 else 0 for i in
df['pub_rec_bankruptcies']]
```

Mapping values and Type Casting Features appropriately

```
df['term'] = df['term'].replace({' 36 months': 36, ' 60 months': 60}).astype(int)
df['loan_status'] = df['loan_status'].replace({'Fully Paid':0, 'Charged
Off':1}).astype(int)
df['initial_list_status'] = df['initial_list_status'].replace({'f': 1, 'w':
0}).astype(int)

years = {'10+ years':10, '4 years':4, '< 1 year':0, '6 years':6, '9 years':9,'2
years':2, '3 years':3,'8 years':8, '7 years':7, '5 years':5, '1 year':1}
df['emp_length']=df['emp_length'].replace(years).astype(int)
```

Deriving impactful Features from present features

```python
# converting the earliest_cr_line column into month and year

df['earliest_cr_line'] = pd.to_datetime(df['earliest_cr_line'])
df['earliest_cr_line_month'] = df['earliest_cr_line'].dt.month
df['earliest_cr_line_year'] = df['earliest_cr_line'].dt.year

df['issue_d'] = pd.to_datetime(df['issue_d'])
df['issue_d_month'] = df['earliest_cr_line'].dt.month
df['issue_d_year'] = df['earliest_cr_line'].dt.year

df.sample(3)
```

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	hor
124376	9000.0	36	12.29	300.18	C	C1	Driver	2	RE
385087	12200.0	36	9.99	393.61	B	B4	DZI Global Inc.	5	RE
13107	10000.0	36	11.99	332.10	B	B5	Registered Nurse	10	MC

Splitting the address column

```python
df[['state', 'zip_code']] = df['address'].apply(lambda x: pd.Series([x[-8:-6],
x[-5:]]))
```

```python
df.info()
df.columns
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 334559 entries, 0 to 396029
Data columns (total 33 columns):
 #   Column                  Non-Null Count   Dtype
---  ------                  --------------   -----
 0   loan_amnt               334559 non-null  float64
 1   term                    334559 non-null  int64
 2   int_rate                334559 non-null  float64
 3   installment             334559 non-null  float64
 4   grade                   334559 non-null  object
 5   sub_grade               334559 non-null  object
 6   emp_title               334559 non-null  object
 7   emp_length              334559 non-null  int64
 8   home_ownership          334559 non-null  object
 9   annual_inc              334559 non-null  float64
 10  verification_status     334559 non-null  object
 11  issue_d                 334559 non-null  datetime64[ns]
 12  loan_status             334559 non-null  int64
 13  purpose                 334559 non-null  object
```

```
 14  title                  334559 non-null  object
 15  dti                    334559 non-null  float64
 16  earliest_cr_line       334559 non-null  datetime64[ns]
 17  open_acc               334559 non-null  float64
 18  pub_rec                334559 non-null  int64
 19  revol_bal              334559 non-null  float64
 20  revol_util             334559 non-null  float64
 21  total_acc              334559 non-null  float64
 22  initial_list_status    334559 non-null  int64
 23  application_type       334559 non-null  object
 24  mort_acc               334559 non-null  int64
 25  pub_rec_bankruptcies   334559 non-null  int64
 26  address                334559 non-null  object
 27  earliest_cr_line_month 334559 non-null  int32
 28  earliest_cr_line_year  334559 non-null  int32
 29  issue_d_month          334559 non-null  int32
 30  issue_d_year           334559 non-null  int32
 31  state                  334559 non-null  object
 32  zip_code               334559 non-null  object
dtypes: datetime64[ns](2), float64(9), int32(4), int64(7), object(11)
memory usage: 81.7+ MB
```

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

Dropping unwanted columns

```
columns = ['sub_grade', 'emp_title', 'issue_d', 'title', 'earliest_cr_line',
'address', 'earliest_cr_line_month', 'earliest_cr_line_year', 'issue_d_month',
'issue_d_year', 'state']
df.drop(columns=columns, inplace=True)
```

Encoding Variables

```
dummies=['grade','home_ownership', 'verification_status', 'purpose',
'application_type', 'zip_code']

data=pd.get_dummies(df,columns=dummies,drop_first=True)
pd.set_option('display.max_columns',None)
pd.set_option('display.max_rows',None)
```

6.0 Model Building

##6.1 Train Test Split

```
from sklearn.model_selection import train_test_split

x = data.drop('loan_status',axis=1)
y = data['loan_status']

# stratify to balance the data during the split
x_train, x_test, y_train, y_test =
train_test_split(x,y,test_size=0.30,stratify=y,random_state=42)

print(f'x_train: {x_train.shape}')
print(f'x_test: {x_test.shape}')
print(f'y_train: {y_train.shape}')
print(f'y_test: {y_test.shape}')
x_train.sample(3, random_state = 42)
```

```
x_train: (234191, 52)
x_test: (100368, 52)
y_train: (234191,)
y_test: (100368,)
```

	loan_amnt	term	int_rate	installment	emp_length	annual_inc	dti	open_acc	pub_rec
262278	16000.0	60	12.99	363.97	10	57000.0	13.98	7.0	0
319937	16625.0	60	18.25	424.43	10	51500.0	33.23	9.0	0
91200	7000.0	36	9.75	225.05	0	63000.0	9.03	6.0	0

```
# Importing stats libraries

from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.metrics import precision_recall_curve
from sklearn.model_selection import train_test_split, KFold, cross_val_score
from sklearn.metrics import (
    accuracy_score, confusion_matrix, classification_report,
    roc_auc_score, roc_curve, auc,
    ConfusionMatrixDisplay, RocCurveDisplay
)
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

```
x_train.columns
```

```
Index(['loan_amnt', 'term', 'int_rate', 'installment', 'emp_length',
       'annual_inc', 'dti', 'open_acc', 'pub_rec', 'revol_bal', 'revol_util',
       'total_acc', 'initial_list_status', 'mort_acc', 'pub_rec_bankruptcies',
       'grade_B', 'grade_C', 'grade_D', 'grade_E', 'grade_F', 'grade_G',
       'home_ownership_MORTGAGE', 'home_ownership_NONE',
       'home_ownership_OTHER', 'home_ownership_OWN', 'home_ownership_RENT',
       'verification_status_Source Verified', 'verification_status_Verified',
       'purpose_credit_card', 'purpose_debt_consolidation',
       'purpose_educational', 'purpose_home_improvement', 'purpose_house',
       'purpose_major_purchase', 'purpose_medical', 'purpose_moving',
       'purpose_other', 'purpose_renewable_energy', 'purpose_small_business',
       'purpose_vacation', 'purpose_wedding', 'application_type_INDIVIDUAL',
       'application_type_JOINT', 'zip_code_05113', 'zip_code_11650',
       'zip_code_22690', 'zip_code_29597', 'zip_code_30723', 'zip_code_48052',
       'zip_code_70466', 'zip_code_86630', 'zip_code_93700'],
      dtype='object')
```

##6.2 Standardization

```
from sklearn.preprocessing import MinMaxScaler
x_train_columns = x_train.columns
scaler = MinMaxScaler()

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

##6.3 Training the Logistic Regression Model

Model 1: Training the unbalanced model after preprocessing.

```
model = LogisticRegression(max_iter=1000)

model.fit(x_train,y_train)
```

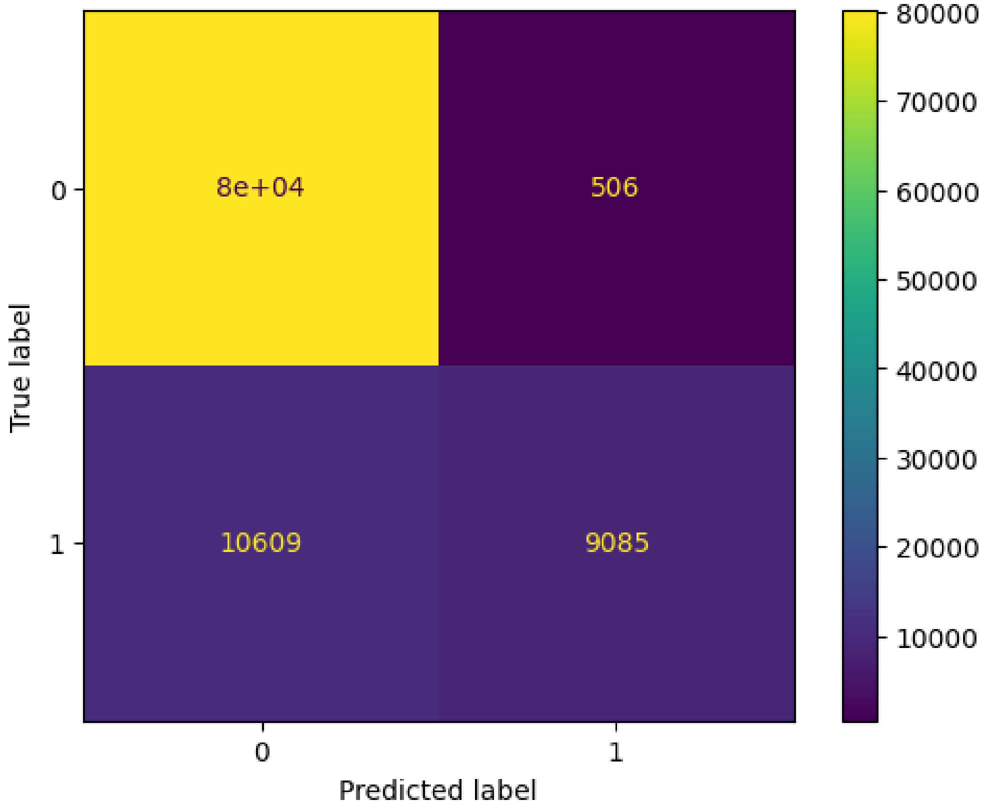
```
LogisticRegression(max_iter=1000)
```

```
y_pred = model.predict(x_test)

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

```
Accuracy of Logistic Regression Classifier on test set: 0.889
```

```
#Plot confusion Matrix
confusion_matrix = confusion_matrix(y_test,y_pred)

print(confusion_matrix)

ConfusionMatrixDisplay(confusion_matrix=confusion_matrix,
display_labels=model.classes_).plot()
```

```
[[80168   506]
 [10609  9085]]
```

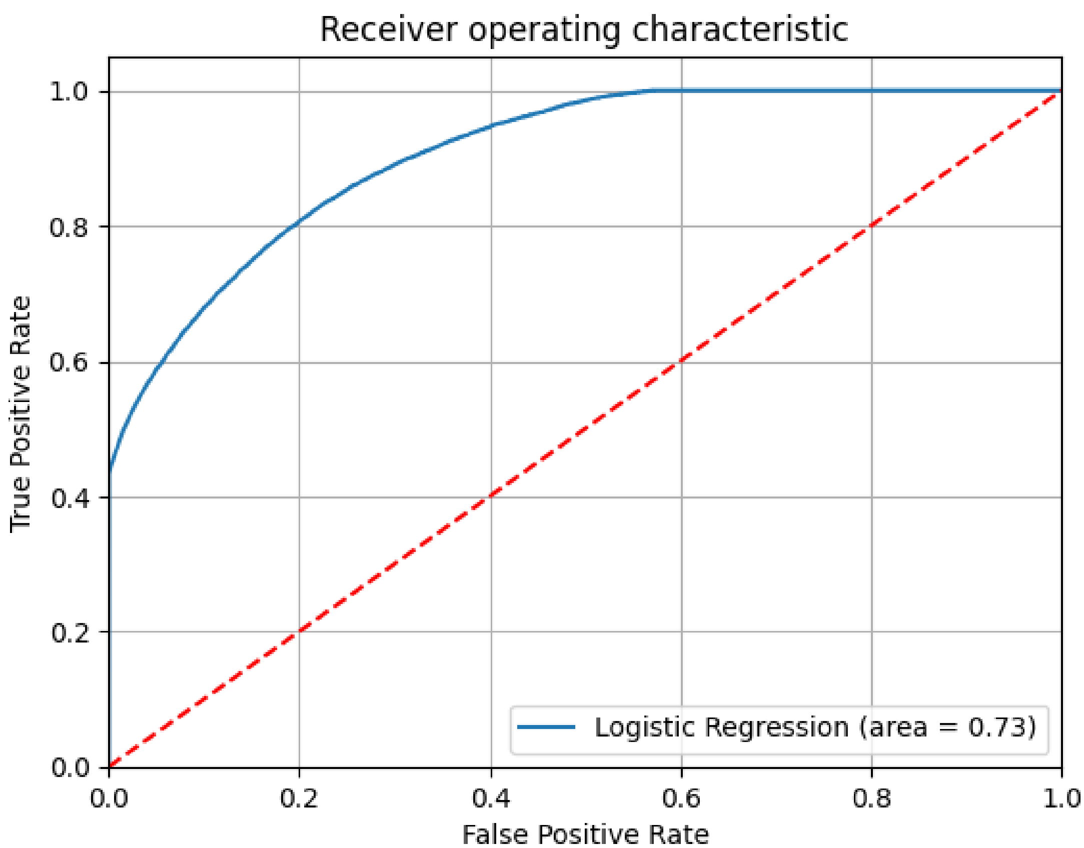



```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.88      0.99      0.94     80674
           1       0.95      0.46      0.62     19694

    accuracy                           0.89    100368
   macro avg       0.92      0.73      0.78    100368
weighted avg       0.90      0.89      0.87    100368
```

ROC Curve

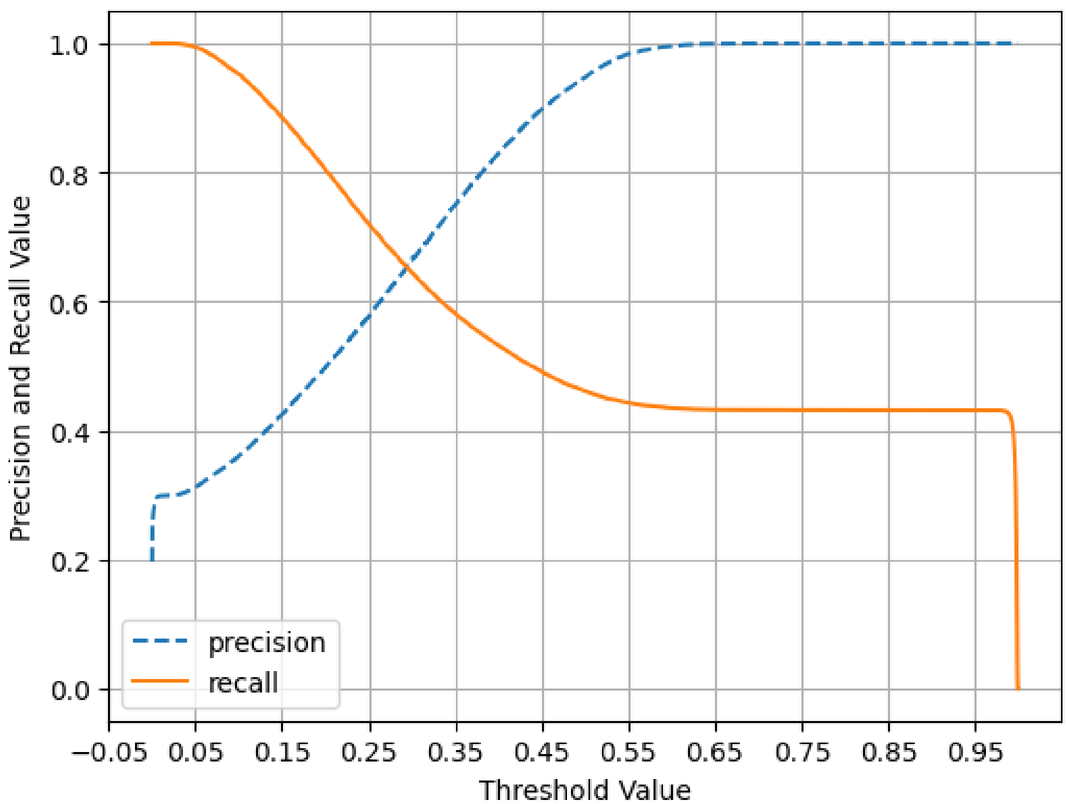
```
logit_roc_auc = roc_auc_score(y_test, model.predict(x_test))

fpr, tpr, thresholds = roc_curve(y_test, model.predict_proba(x_test)[:,1])


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



Precision-Recall Curve

```
precisions, recalls, thresholds = precision_recall_curve(y_test,
model.predict_proba(x_test)[:, 1])
```

```
threshold_boundary = thresholds.shape[0]

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

# Plot recall
plt.plot(thresholds, recalls[0:threshold_boundary], label='recall')

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

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



Model Interpretability

```python
model.score(x_train, y_train)
```

0.8889666981224726

```python
from sklearn.metrics import f1_score
f1 = f1_score(y_test,y_pred)
f1
```

0.6204541574184737

```python
len((model.coef_)[0])
```

52

```python
len(x.columns)
```

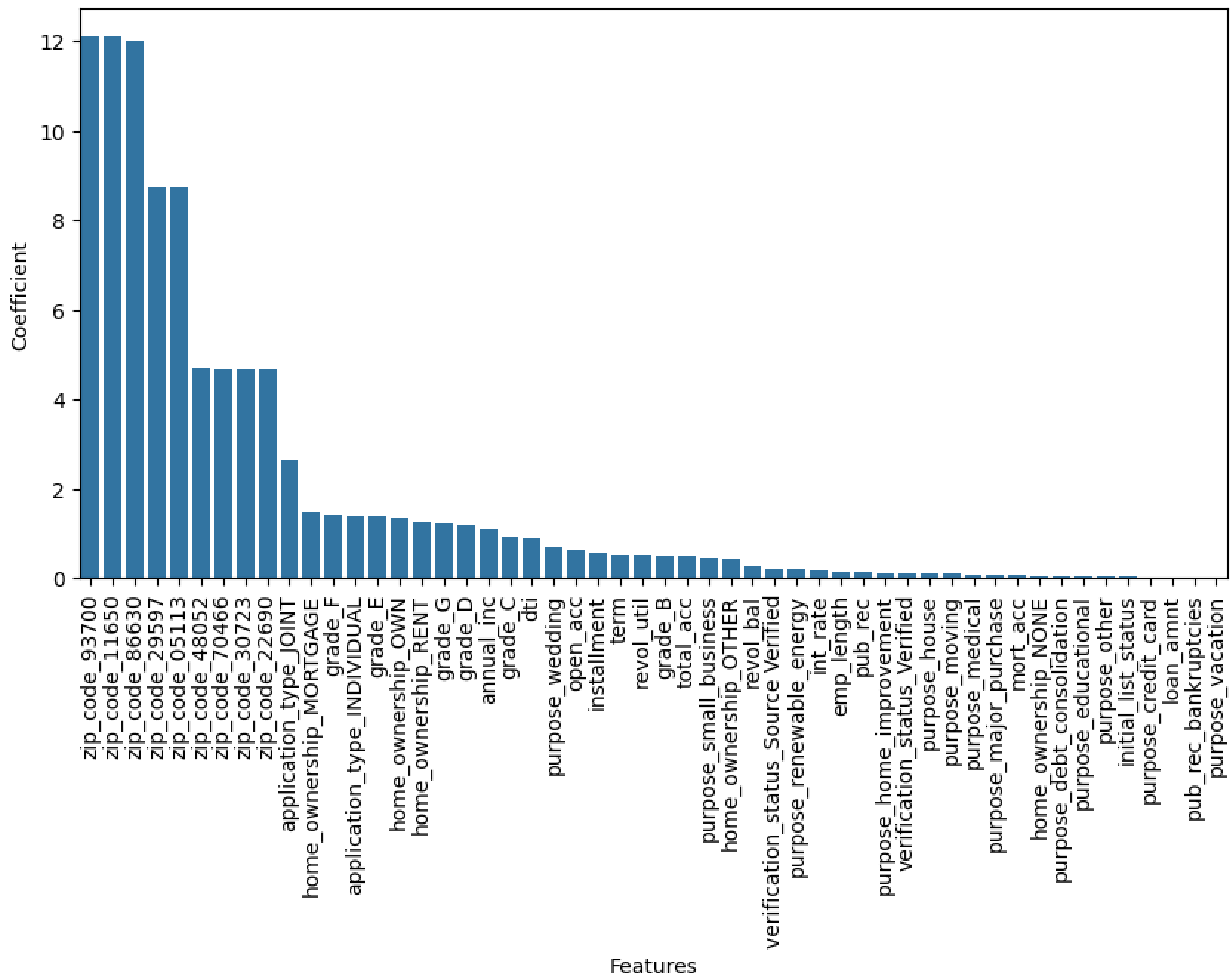
52

```python
imp = pd.DataFrame(list(zip(x.columns,np.abs(model.coef_[0]))),
                   columns=['feature', 'coeff'])
imp = imp.sort_values(by='coeff', ascending=False)
imp
```

	feature	coeff
51	zip_code_93700	12.120141
44	zip_code_11650	12.110739
50	zip_code_86630	12.014185
46	zip_code_29597	8.746250
43	zip_code_05113	8.742120
48	zip_code_48052	4.696756
49	zip_code_70466	4.665408
47	zip_code_30723	4.665069
45	zip_code_22690	4.652905
42	application_type_JOINT	2.655026
21	home_ownership_MORTGAGE	1.479970
19	grade_F	1.444820
41	application_type_INDIVIDUAL	1.410832
18	grade_E	1.382377
24	home_ownership_OWN	1.350490
25	home_ownership_RENT	1.264863
20	grade_G	1.236923
17	grade_D	1.185682
5	annual_inc	1.097888
16	grade_C	0.929183

	feature	coeff
6	dti	0.891909
40	purpose_wedding	0.686778
7	open_acc	0.634078
3	installment	0.565221
1	term	0.536095
10	revol_util	0.532728
15	grade_B	0.509978
11	total_acc	0.507791
38	purpose_small_business	0.478483
23	home_ownership_OTHER	0.423273
9	revol_bal	0.273116
26	verification_status_Source Verified	0.207754
37	purpose_renewable_energy	0.190945
2	int_rate	0.169088
4	emp_length	0.128751
8	pub_rec	0.127149
31	purpose_home_improvement	0.119284
27	verification_status_Verified	0.108641
32	purpose_house	0.105598
35	purpose_moving	0.090243
34	purpose_medical	0.078401
33	purpose_major_purchase	0.068095
13	mort_acc	0.057324
22	home_ownership_NONE	0.056405
29	purpose_debt_consolidation	0.037620
30	purpose_educational	0.035775
36	purpose_other	0.025068
12	initial_list_status	0.024673
28	purpose_credit_card	0.023779
0	loan_amnt	0.021607
14	pub_rec_bankruptcies	0.013940
39	purpose_vacation	0.001112

```
plt.figure(figsize=(10,5))
sns.barplot(x=imp.feature, y='coeff', data=imp)
plt.xlabel('Features')
plt.ylabel('Coefficient')
plt.xticks(rotation=90)
plt.show()
```




```
x.columns[np.argmax(np.abs(model.coef_))]
```

```
'zip_code_93700'
```

```
x.columns[np.argmin(np.abs(model.coef_))]
```

```
'purpose_vacation'
```

So, - `year` is most important feature, - while `manual` is the least important.

Validation

```
x=scaler.fit_transform(x)

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

```
Cross Validation accuracy : 0.889
```

Model 2: Re-Training after oversampling the imbalanced data with SMOTE

```
from imblearn.over_sampling import SMOTE

sm = SMOTE(random_state=42)
x_sm ,y_sm = sm.fit_resample(x_train,y_train)
```

```
print('Before SMOTE')
print(y_train.value_counts())
print('\n')
print('After Oversampling')
print(y_sm.value_counts())
```

```
Before SMOTE
loan_status
0    188240
1     45951
Name: count, dtype: int64


After Oversampling
loan_status
0    188240
1    188240
Name: count, dtype: int64
```

```
lr1 = LogisticRegression(max_iter=1000)
lr1.fit(x_sm, y_sm)
```

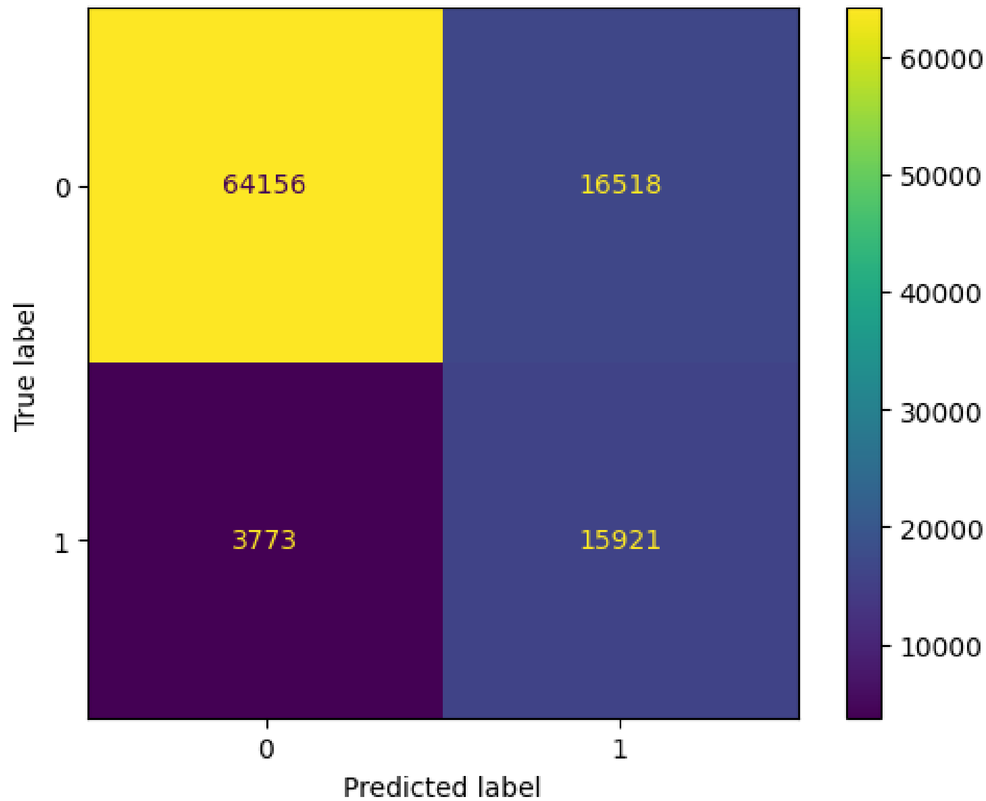
```
LogisticRegression(max_iter=1000)
```

```
y_pred = lr1.predict(x_test)
print('Accuracy of Logistic Regression Classifier on test set:
{:.3f}'.format(lr1.score(x_test, y_test)))
```

```
Accuracy of Logistic Regression Classifier on test set: 0.798
```

```
#Plot confusion Matrix
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
cm = confusion_matrix(y_test, y_pred)

print(cm)

ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=lr1.classes_).plot()
```

```
[[64156 16518]
 [ 3773 15921]]
```



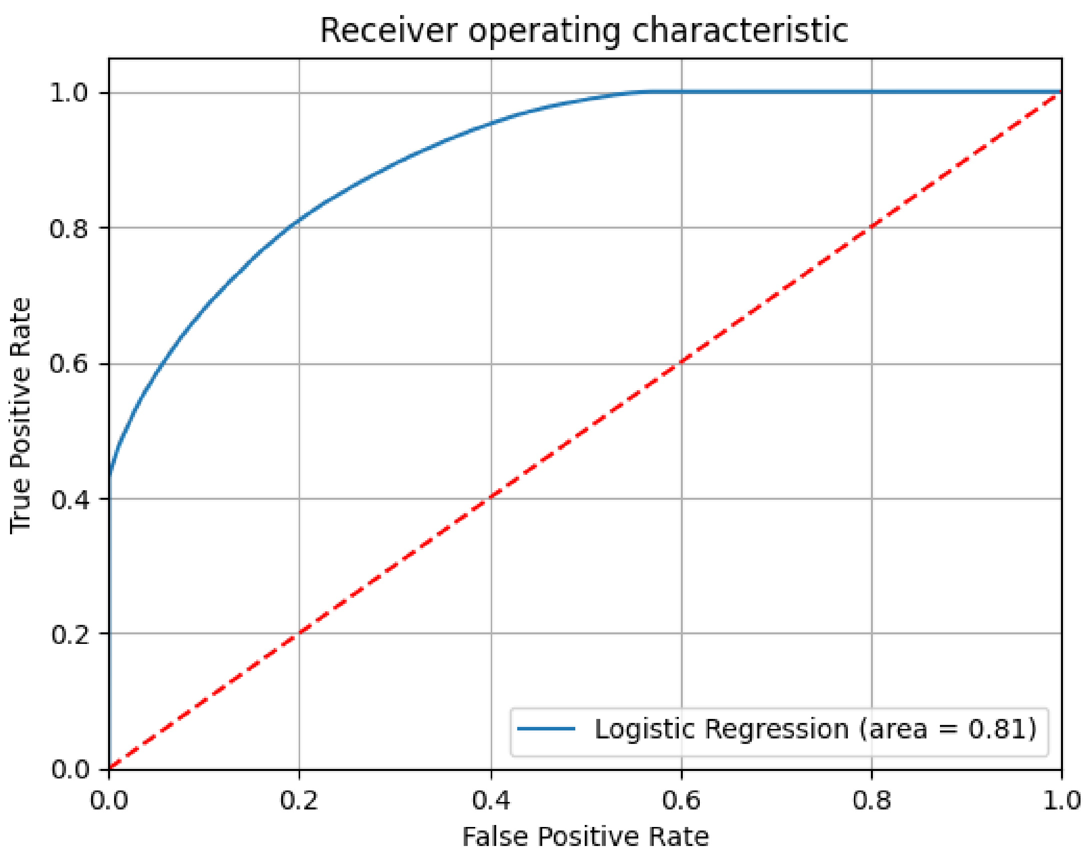

```python
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.94      0.80      0.86     80674
           1       0.49      0.81      0.61     19694

    accuracy                           0.80    100368
   macro avg       0.72      0.80      0.74    100368
weighted avg       0.86      0.80      0.81    100368
```

ROC Curve

```python
logit_roc_auc = roc_auc_score(y_sm, lr1.predict(x_sm))

fpr, tpr, thresholds = roc_curve(y_sm, lr1.predict_proba(x_sm)[:,1])


plt.figure()
plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc)
```

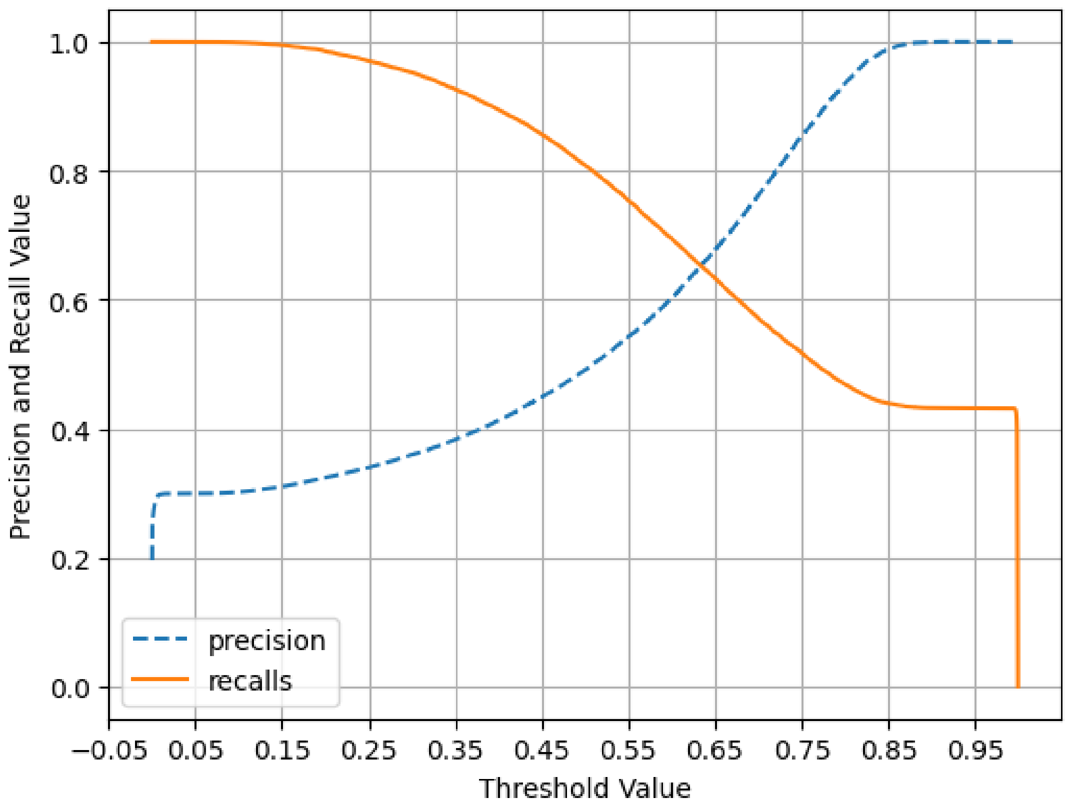
```
plt.plot([0, 1], [0, 1], 'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.grid()
plt.show()
```



Precision Recall Curve

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

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

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

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

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



Model Interpretability

```
lr1.score(x_sm, y_sm)
```

```
0.8050148746281343
```

```
from sklearn.metrics import f1_score
f1 = f1_score(y_test,y_pred)
f1
```

```
0.6107839564191587
```

```python
len((lr1.coef_)[0])
```

52

```python
imp = pd.DataFrame(list(zip(x_train_columns,np.abs(lr1.coef_[0]))),
                   columns=['feature', 'coeff'])
imp = imp.sort_values(by='coeff', ascending=False)
imp
```

	feature	coeff
44	zip_code_11650	13.713803
51	zip_code_93700	13.706281
50	zip_code_86630	13.515730
46	zip_code_29597	11.478334
43	zip_code_05113	11.476029
48	zip_code_48052	5.917488
47	zip_code_30723	5.888397
45	zip_code_22690	5.885856
49	zip_code_70466	5.885732
42	application_type_JOINT	2.828304
41	application_type_INDIVIDUAL	1.451534
19	grade_F	1.442101
5	annual_inc	1.436967
21	home_ownership_MORTGAGE	1.435192
18	grade_E	1.386479
24	home_ownership_OWN	1.330601
25	home_ownership_RENT	1.244838
17	grade_D	1.214897
20	grade_G	1.143486
6	dti	0.968973
16	grade_C	0.961118
40	purpose_wedding	0.950262
7	open_acc	0.883222
10	revol_util	0.795406
11	total_acc	0.684264
38	purpose_small_business	0.588705
23	home_ownership_OTHER	0.583070
3	installment	0.581524
15	grade_B	0.557513
1	term	0.556095
9	revol_bal	0.500589
32	purpose_house	0.347586
31	purpose_home_improvement	0.268180
8	pub_rec	0.212611
26	verification_status_Source Verified	0.197600

	feature	coeff
30	purpose_educational	0.191573
36	purpose_other	0.188760
29	purpose_debt_consolidation	0.182630
35	purpose_moving	0.137680
2	int_rate	0.131791
28	purpose_credit_card	0.116827
33	purpose_major_purchase	0.115089
4	emp_length	0.111918
0	loan_amnt	0.111221
27	verification_status_Verified	0.090435
14	pub_rec_bankruptcies	0.089045
37	purpose_renewable_energy	0.084916
39	purpose_vacation	0.081896
34	purpose_medical	0.048831
22	home_ownership_NONE	0.042918
13	mort_acc	0.038634
12	initial_list_status	0.006937

####**Comparitive Model Analysis between model 1 and 2**

Key Metric Comparisons

- **True Positive Rate (Recall)**
  - Previous Model: 46.13% New Model: 80.84%
- **True Negative Rate (Specificity)**
  - Previous Model: 99.37% New Model: 79.52%
- **False Positive Rate (FPR)**
  - Previous Model: 0.63% New Model: 20.48%
- **False Negative Rate (FNR)**
  - Previous Model: 53.87% New Model: 19.16%

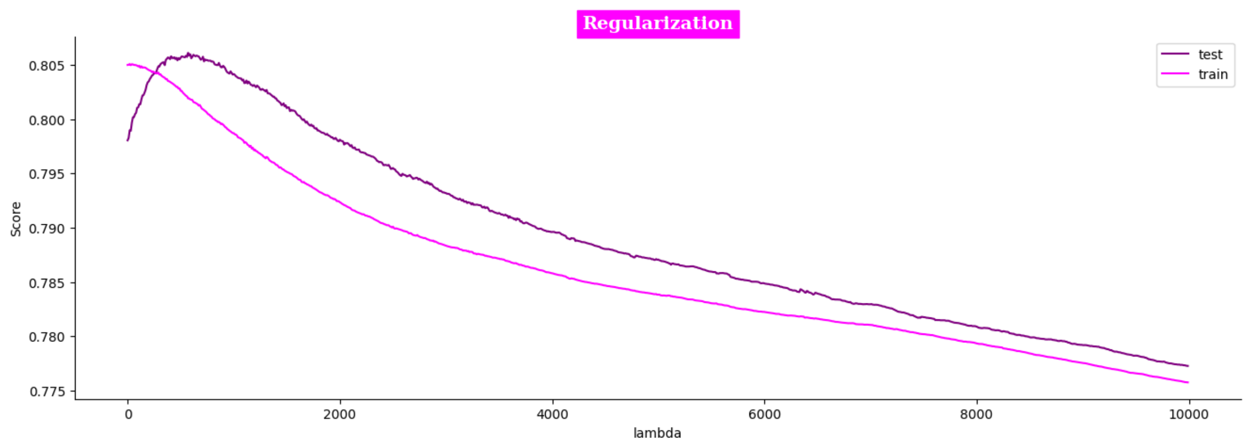
Key Observations

- **Recall (TPR):**
  - The new model is far better at detecting positive cases (80.84% vs. 46.13%). If detecting risky loans is critical, this model is more suitable.
- **Specificity (TNR):**
  - The new model loses its ability to correctly classify negative cases, dropping from 99.37% to 79.52%. This could lead to operational inefficiencies or unnecessary restrictions for borrowers.

Model 3: Retraining the model after Regularization

```
#Try with different regularization factor lamda and choose the best to build the
model

lamb = np.arange(0.01, 10000, 10)

train_scores = []
test_scores = []

for lam in lamb:
    model = LogisticRegression(C = 1/lam)
    model.fit(x_sm, y_sm)

    tr_score = model.score(x_sm, y_sm)
    te_score = model.score(x_test, y_test)

    train_scores.append(tr_score)
    test_scores.append(te_score)
```

```
#Plot the train and test scores with respect lambda values i.e. regularization
factors
ran = np.arange(0.01, 10000, 10)
plt.figure(figsize=(16,5))
sns.lineplot(x=ran,y=test_scores,color='purple',label='test')
sns.lineplot(x=ran,y=train_scores,color='magenta',label='train')
plt.title('Regularization',fontsize=14,fontfamily='serif',fontweight='bold'
,backgroundcolor='magenta',color='w')
plt.xlabel("lambda")
plt.ylabel("Score")
sns.despine()
plt.show()
```



```
#Check the index of best test score and the check the best test score
a = np.argmax(test_scores)
print(np.argmax(test_scores))
print(test_scores[np.argmax(test_scores)])
```

```
57
0.8061135023114937
```

```
#Calculate the best lambda value based on the index of best test score

best_lamb = 0.01 + (10*a)
best_lamb
```

```
570.01
```

```


#Fit the model using best lambda

reg_model = LogisticRegression(C=1/best_lamb)
reg_model.fit(x_sm, y_sm)
```

```
LogisticRegression(C=0.0017543551867511096)
```

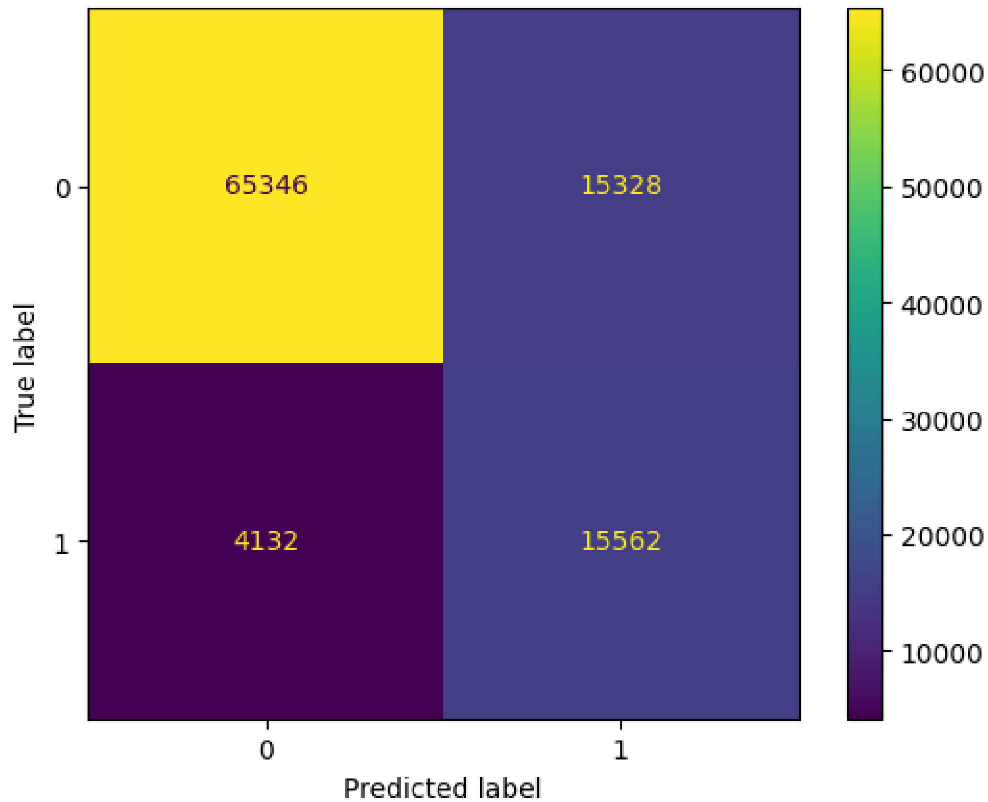
```
y_reg_pred = reg_model.predict(x_test)
y_reg_pred_proba = reg_model.predict_proba(x_test)
```

```
#Print model score

print(f'Logistic Regression Model Score with best lambda: ',end='')
print(round(reg_model.score(x_test, y_test)*100,2),'%')
```

```
Logistic Regression Model Score with best lambda: 80.61 %
```

```
#Plot confusion Matrix
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
cm_reg = confusion_matrix(y_test, y_reg_pred)

print(cm_reg)

ConfusionMatrixDisplay(confusion_matrix=cm_reg,
display_labels=reg_model.classes_).plot()
```

```
[[65346 15328]
 [ 4132 15562]]
```



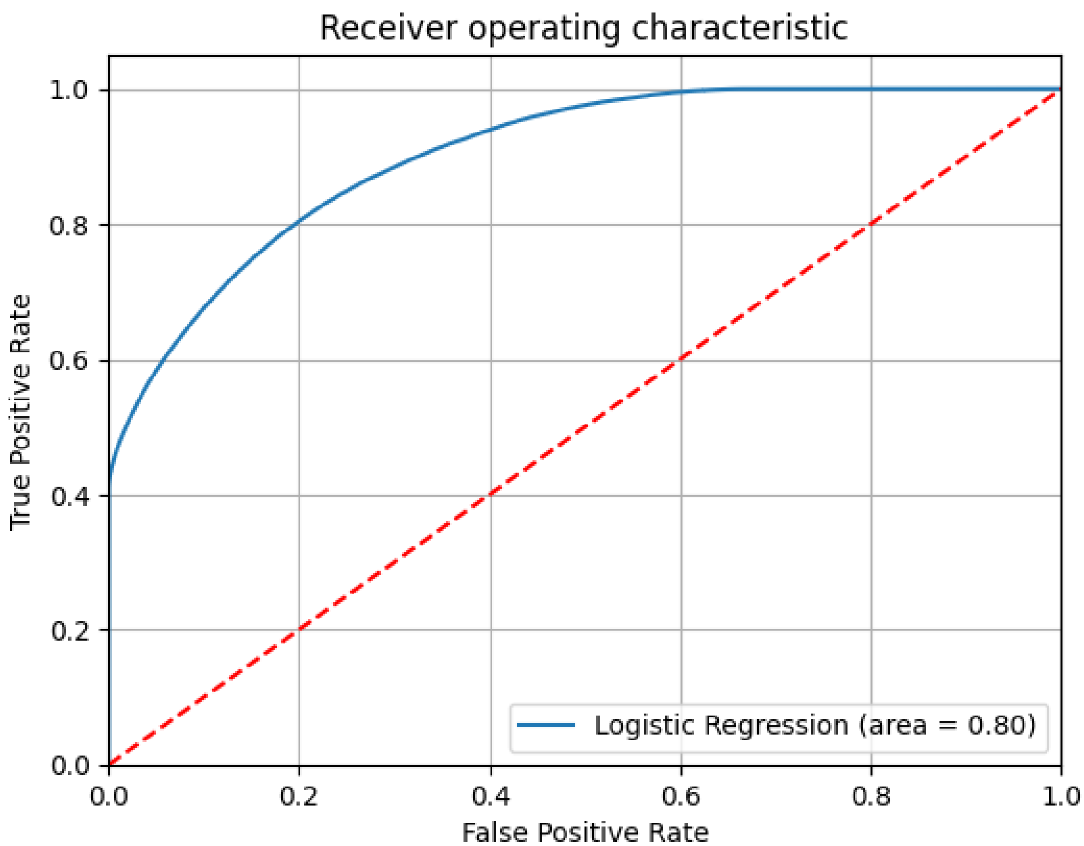

```
print(classification_report(y_test,y_reg_pred))
```

```
              precision    recall  f1-score   support

           0       0.94      0.81      0.87     80674
           1       0.50      0.79      0.62     19694

    accuracy                           0.81    100368
   macro avg       0.72      0.80      0.74    100368
weighted avg       0.85      0.81      0.82    100368
```

ROC Curve

```
logit_roc_auc = roc_auc_score(y_sm, reg_model.predict(x_sm))

fpr, tpr, thresholds = roc_curve(y_sm, reg_model.predict_proba(x_sm)[:,1])


plt.figure()
plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc)
plt.plot([0, 1], [0, 1], 'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
```

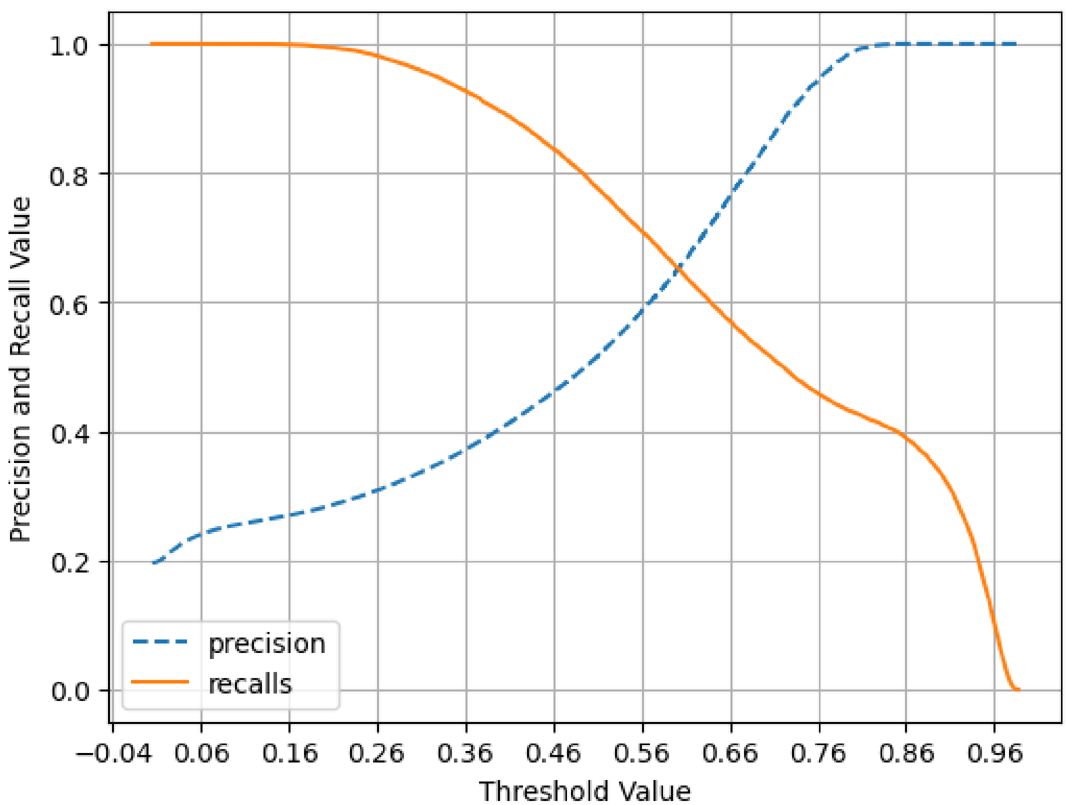
```
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.grid()
plt.show()
```



Precision Recall Curve

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

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

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

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

precision_recall_curve_plot(y_test, reg_model.predict_proba(x_test)[:,1])
```



Model Interpretability

```
reg_model.score(x_sm, y_sm)
```

0.8019151083722907

```
from sklearn.metrics import f1_score
f1 = f1_score(y_test,y_reg_pred)
f1
```

0.6152933733987032

```
len((model.coef_)[0])
```

52

```
imp = pd.DataFrame(list(zip(x_train_columns,np.abs(reg_model.coef_[0]))),
                   columns=['feature', 'coeff'])
imp = imp.sort_values(by='coeff', ascending=False)
imp
```

	feature	coeff
51	zip_code_93700	3.544363
44	zip_code_11650	3.544143
50	zip_code_86630	3.499020
46	zip_code_29597	1.985616
43	zip_code_05113	1.985040
48	zip_code_48052	1.053550
47	zip_code_30723	1.027828
49	zip_code_70466	1.024588
45	zip_code_22690	1.023873
5	annual_inc	0.924410
2	int_rate	0.924240
6	dti	0.808916
18	grade_E	0.576295
17	grade_D	0.536851
1	term	0.488290
10	revol_util	0.478907
19	grade_F	0.449085
16	grade_C	0.420935
7	open_acc	0.378660
11	total_acc	0.338513
38	purpose_small_business	0.256389
3	installment	0.239497
9	revol_bal	0.230058
40	purpose_wedding	0.220220
26	verification_status_Source Verified	0.184751
15	grade_B	0.162290
0	loan_amnt	0.144773
32	purpose_house	0.125965
8	pub_rec	0.123842
31	purpose_home_improvement	0.119976
42	application_type_JOINT	0.118645
21	home_ownership_MORTGAGE	0.112482
4	emp_length	0.105678
27	verification_status_Verified	0.101759
29	purpose_debt_consolidation	0.099941
13	mort_acc	0.095990
36	purpose_other	0.075369
39	purpose_vacation	0.073476

	feature	coeff
20	grade_G	0.072818
41	application_type_INDIVIDUAL	0.072133
25	home_ownership_RENT	0.068184
35	purpose_moving	0.034857
28	purpose_credit_card	0.030419
23	home_ownership_OTHER	0.019537
33	purpose_major_purchase	0.015906
12	initial_list_status	0.011908
30	purpose_educational	0.011051
34	purpose_medical	0.010513
14	pub_rec_bankruptcies	0.007532
22	home_ownership_NONE	0.002707
37	purpose_renewable_energy	0.001314
24	home_ownership_OWN	0.001086

7.0 Model Comparison of 1, 2 and 3

Model	Precision (Class 1)	Recall (Class 1)	F1-score (Class 1)	Accuracy	ROC AUC
Model 1	0.95	0.46	0.62	0.89	0.73
Model 2 (SMOTE)	0.49	0.81	0.61	0.8	0.81
Model 3 (Regularization)	0.5	0.79	0.62	0.81	0.8

Figure 1: image.png

Trade-off Analysis

- **Model 1:**
  - Strengths: High accuracy and precision for the majority class (class 0).
  - Weaknesses: Low recall for the minority class (class 1), indicating a high number of false negatives. This model might be biased towards the majority class due to class imbalance.
- **Model 2 (SMOTE):**
  - Strengths: Improved recall for the minority class (class 1) compared to Model 1. This suggests that SMOTE effectively addressed the class imbalance issue.
  - Weaknesses: Lower precision for the minority class, indicating a higher number of false positives. Accuracy also decreased compared to Model 1.
- **Model 3 (Regularization):**
  - Strengths: Similar performance to Model 2 in terms of recall for the minority class.
  - Weaknesses: Slightly lower precision for the minority class compared to Model 2.

Comprehensive outlook on models - There is no significant difference in the performance metrics of model 2 and 3. So we can go ahead with Model 2. - The model 2 prioritizes minimizing false negatives at the expense of increasing false positives. Whether this is acceptable depends on the business's tolerance for false positives.

Which Model is Better? - If recall is critical (e.g., identifying risky loans to reduce default rates), the model 2 is better. - If specificity and minimizing false positives are more important (e.g., ensuring good customer experience by not falsely flagging loans), the model 1 is preferable.

8.0 Insights:

- **Class Imbalance:** The dataset exhibits significant class imbalance, with the majority class (class 0) dominating predictions. This imbalance skews model performance and reduces the ability to detect minority class (class 1) instances effectively.
- **SMOTE Impact:** SMOTE addressed class imbalance by generating synthetic samples for the minority class, improving recall but increasing false positives.
- **Regularization Impact:** Regularization helped mitigate overfitting but did not significantly improve recall compared to SMOTE.
- **Model Trade-off:** The new model (using SMOTE or regularization) achieved better recall but at the cost of increased false positives, highlighting the inherent trade-off between recall and specificity.
- **Business Implications:** The choice of model should depend on whether false positives (e.g., unnecessary loan rejections) or false negatives (e.g., missed risky loans) have a greater financial or operational impact.

Recommendations:

- Explore creating or transforming features to provide the model with better predictive power.
- Incorporate feedback from end-users or collaborate with domain experts to better understand the real-world implications of false positives and false negatives.
- Regularly monitor model performance in production and retrain it with updated data to prevent performance degradation over time.
- Collect more relevant data to reduce the influence of majority classes and set up automated alerts for performance drops.
- Maintain thorough documentation of changes, updates, and performance metrics for transparency and accountability.