In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
from scipy import stats
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.metrics import precision_recall_curve
from sklearn.model_selection import train_test_split, KFold, cross_val_score
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import (
    accuracy_score, confusion_matrix, classification_report,
    roc_auc_score, roc_curve, auc,
    ConfusionMatrixDisplay, RocCurveDisplay
)
from statsmodels.stats.outliers_influence import variance_inflation_factor
from imblearn.over_sampling import SMOTE
```

In [2]:
```
df_LOR = pd.read_csv('/Users/Ramv/Downloads/logistic_regression.csv')
df_LOR
```

Out[2]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_len
0	10000.0	36 months	11.44	329.48	B	B4	Marketing	10+ ye
1	8000.0	36 months	11.99	265.68	B	B5	Credit analyst	4 ye
2	15600.0	36 months	10.49	506.97	B	B3	Statistician	< 1 y
3	7200.0	36 months	6.49	220.65	A	A2	Client Advocate	6 ye
4	24375.0	60 months	17.27	609.33	C	C5	Destiny Management Inc.	9 ye
...	...	...	...	...	...	...	...	
396025	10000.0	60 months	10.99	217.38	B	B4	licensed bankere	2 ye
396026	21000.0	36 months	12.29	700.42	C	C1	Agent	5 ye
396027	5000.0	36 months	9.99	161.32	B	B1	City Carrier	10+ ye
396028	21000.0	60 months	15.31	503.02	C	C2	Gracon Services, Inc	10+ ye
396029	2000.0	36 months	13.61	67.98	C	C2	Internal Revenue Service	10+ ye

396030 rows × 27 columns

In [3]:

```
df_LOR.isnull().sum()
```

Out[3]:
```
loan_amnt                   0
term                        0
int_rate                    0
installment                 0
grade                       0
sub_grade                   0
emp_title               22927
emp_length              18301
home_ownership              0
annual_inc                  0
verification_status         0
issue_d                     0
loan_status                 0
purpose                     0
title                    1755
dti                         0
earliest_cr_line            0
open_acc                    0
pub_rec                     0
revol_bal                   0
revol_util                276
total_acc                   0
initial_list_status         0
application_type            0
mort_acc                37795
pub_rec_bankruptcies      535
address                     0
dtype: int64
```

In [4]:
```
df_LOR.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 396030 entries, 0 to 396029
Data columns (total 27 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   loan_amnt             396030 non-null  float64
 1   term                  396030 non-null  object 
 2   int_rate              396030 non-null  float64
 3   installment           396030 non-null  float64
 4   grade                 396030 non-null  object 
 5   sub_grade             396030 non-null  object 
 6   emp_title             373103 non-null  object 
 7   emp_length            377729 non-null  object 
 8   home_ownership        396030 non-null  object 
 9   annual_inc            396030 non-null  float64
 10  verification_status   396030 non-null  object 
 11  issue_d               396030 non-null  object 
 12  loan_status           396030 non-null  object 
 13  purpose               396030 non-null  object 
 14  title                 394275 non-null  object 
 15  dti                   396030 non-null  float64
 16  earliest_cr_line      396030 non-null  object 
 17  open_acc              396030 non-null  float64
 18  pub_rec               396030 non-null  float64
 19  revol_bal             396030 non-null  float64
 20  revol_util            395754 non-null  float64
 21  total_acc             396030 non-null  float64
 22  initial_list_status   396030 non-null  object 
 23  application_type      396030 non-null  object 
 24  mort_acc              358235 non-null  float64
 25  pub_rec_bankruptcies  395495 non-null  float64
 26  address               396030 non-null  object 
dtypes: float64(12), object(15)
memory usage: 81.6+ MB
```

In [5]:
```
df_LOR.describe()
```

Out[5]:

	loan_amnt	int_rate	installment	annual_inc	dti
count	396030.000000	396030.000000	396030.000000	3.960300e+05	396030.000000	396(
mean	14113.888089	13.639400	431.849698	7.420318e+04	17.379514
std	8357.441341	4.472157	250.727790	6.163762e+04	18.019092
min	500.000000	5.320000	16.080000	0.000000e+00	0.000000
25%	8000.000000	10.490000	250.330000	4.500000e+04	11.280000
50%	12000.000000	13.330000	375.430000	6.400000e+04	16.910000
75%	20000.000000	16.490000	567.300000	9.000000e+04	22.980000
max	40000.000000	30.990000	1533.810000	8.706582e+06	9999.000000

In [6]:
```
df_LOR.isna().sum()*100/df_LOR.shape[0]
```

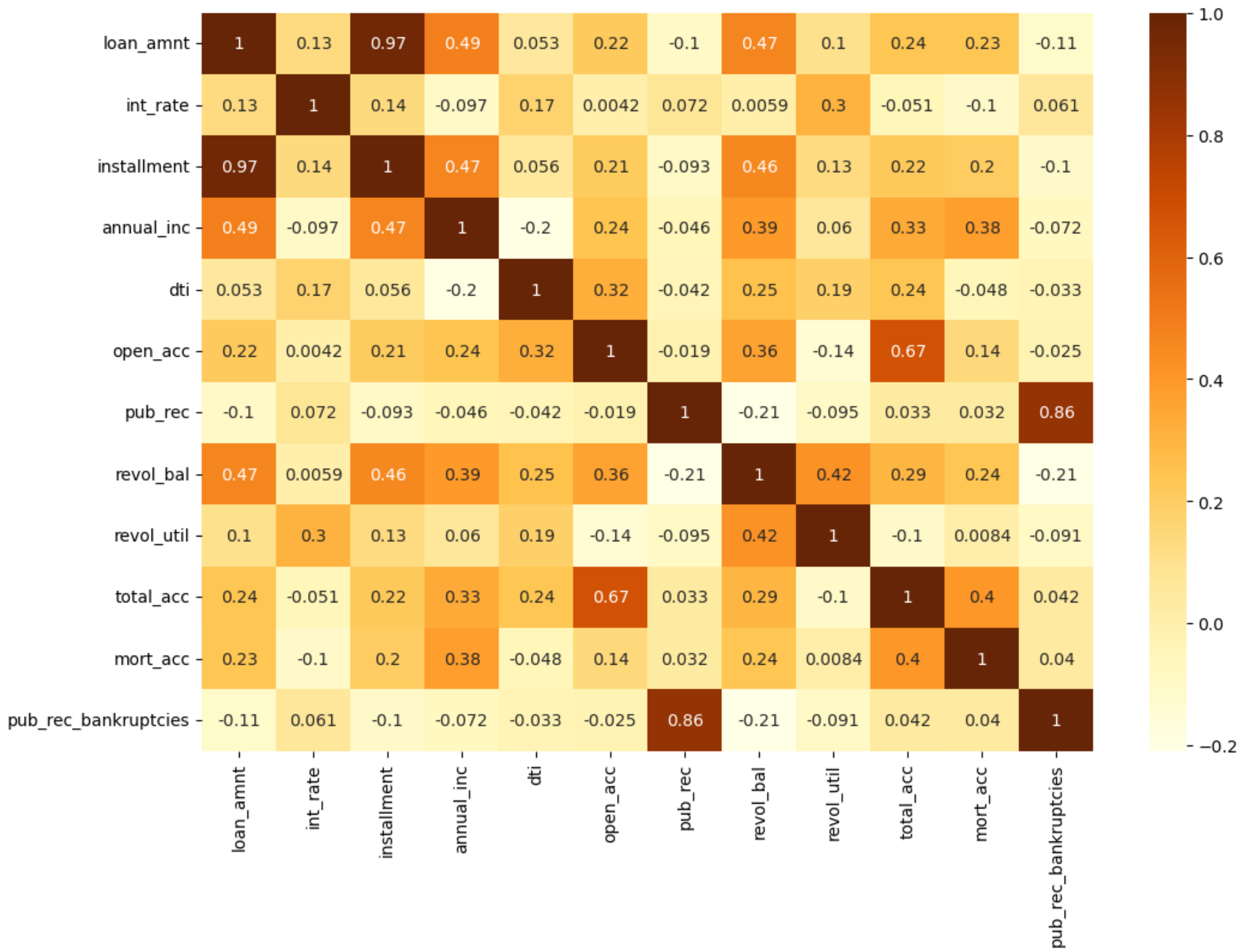
Out[6]:
```
loan_amnt               0.000000
term                    0.000000
int_rate                0.000000
installment             0.000000
grade                   0.000000
sub_grade               0.000000
emp_title               5.789208
emp_length              4.621115
home_ownership          0.000000
annual_inc              0.000000
verification_status     0.000000
issue_d                 0.000000
loan_status             0.000000
purpose                 0.000000
title                   0.443148
dti                     0.000000
earliest_cr_line        0.000000
open_acc                0.000000
pub_rec                 0.000000
revol_bal               0.000000
revol_util              0.069692
total_acc               0.000000
initial_list_status     0.000000
application_type        0.000000
mort_acc                9.543469
pub_rec_bankruptcies    0.135091
address                 0.000000
dtype: float64
```

In [7]:
```
df_LOR.loan_status.value_counts(normalize=True)*100
```

Out[7]:
```
Fully Paid     80.387092
Charged Off    19.612908
Name: loan_status, dtype: float64
```

Correlation check

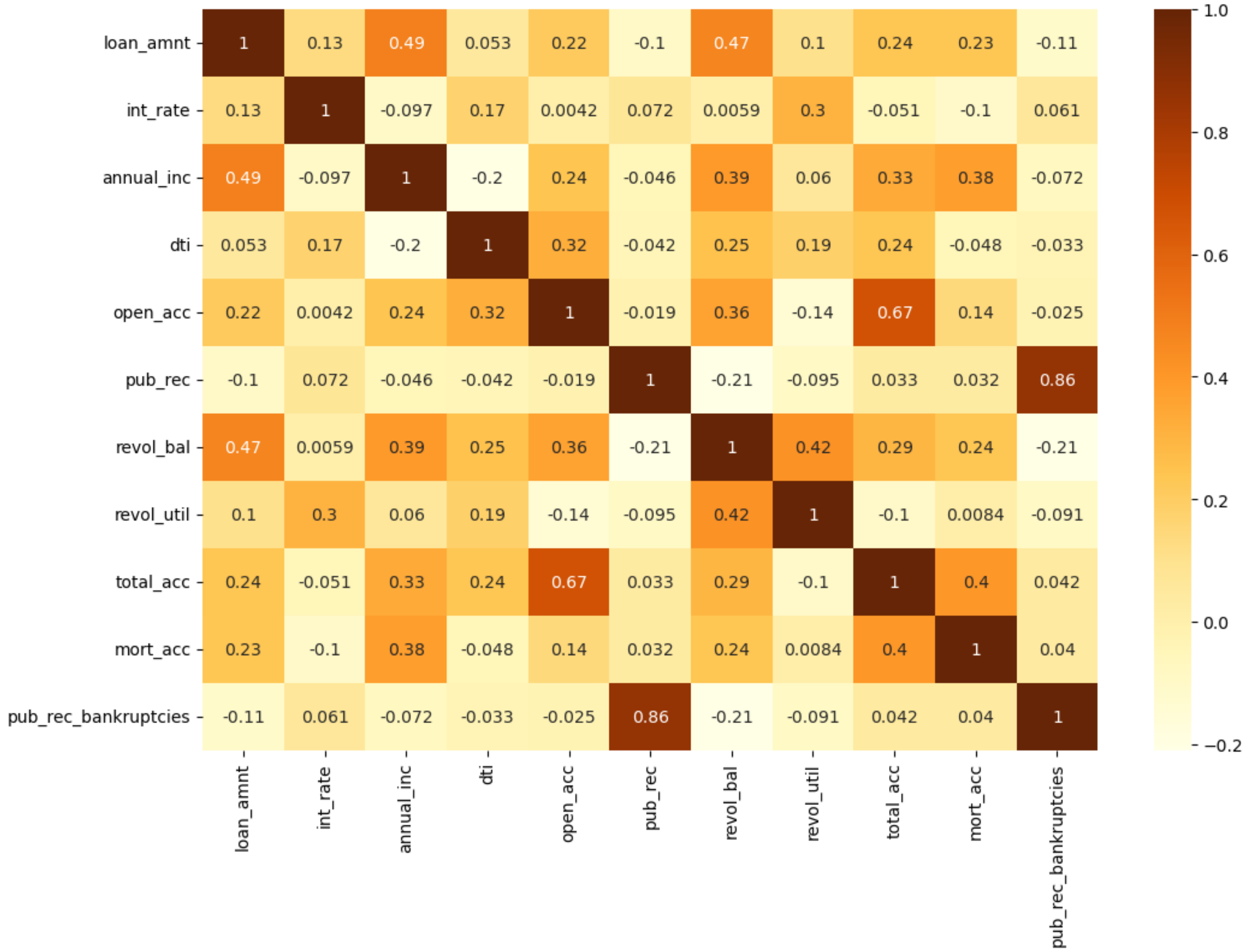
In [8]:
```
import warnings
warnings.filterwarnings("ignore")
plt.figure(figsize=(12,8))
sns.heatmap(df_LOR.corr(method='spearman'),annot=True,cmap='YlOrBr')
plt.show()
```



Observation: There is an almost perfect correlation between "loan_amnt" & "installment" features. We can drop the "installment" feature

In [9]:
```
df_LOR.drop(columns=['installment'],axis=1,inplace=True)
```

In [10]:
```
plt.figure(figsize=(12, 8))
sns.heatmap(df_LOR.corr(method='spearman'), annot=True, cmap='YlOrBr')
plt.show()
```



Data Exploration

loan_status vs loan_amnt

In [11]:
```
df_LOR.groupby(by='loan_status')['loan_amnt'].describe()
```

Out[11]:

loan_status	count	mean	std	min	25%	50%	75%	ma
Charged Off	77673.0	15126.300967	8505.090557	1000.0	8525.0	14000.0	20000.0	40000.0
Fully Paid	318357.0	13866.878771	8302.319699	500.0	7500.0	12000.0	19225.0	40000.0

home_ownership

In [12]:
```
df_LOR['home_ownership'].value_counts()
```

Out[12]:
```
MORTGAGE    198348
RENT        159790
OWN          37746
OTHER          112
NONE            31
ANY              3
Name: home_ownership, dtype: int64
```

Combining the minority classes such as 'None' & 'Any' as 'OTHER'

In [13]:
```
df_LOR.loc[(df_LOR.home_ownership == 'NONE')| (df_LOR.home_ownership == 'ANY
df_LOR['home_ownership'].value_counts()
```

Out[13]:
```
MORTGAGE    198348
RENT        159790
OWN          37746
OTHER          146
Name: home_ownership, dtype: int64
```

checking loan status distribution for the home_ownership category 'OTHER'

In [14]:
```
# Checking the distribution of 'Other'
df_LOR.loc[df_LOR['home_ownership']=='OTHER','loan_status'].value_counts()
```

Out[14]:
```
Fully Paid     123
Charged Off     23
Name: loan_status, dtype: int64
```

'issue_d' data conversion to date_time format

In [15]:
```
df_LOR['issue_d']=pd.to_datetime(df_LOR['issue_d'])
df_LOR['earliest_cr_line']=pd.to_datetime(df_LOR['earliest_cr_line'])
```

In [16]:
```
df_LOR['issue_d']
```

Out[16]:
```
0         2015-01-01
1         2015-01-01
2         2015-01-01
3         2014-11-01
4         2013-04-01
             ...
396025   2015-10-01
396026   2015-02-01
396027   2013-10-01
396028   2012-08-01
396029   2010-06-01
Name: issue_d, Length: 396030, dtype: datetime64[ns]
```

In [17]:
```
df_LOR['earliest_cr_line']
```

Out[17]:
```
0         1990-06-01
1         2004-07-01
2         2007-08-01
3         2006-09-01
4         1999-03-01
             ...
396025    2004-11-01
396026    2006-02-01
396027    1997-03-01
396028    1990-11-01
396029    1998-09-01
Name: earliest_cr_line, Length: 396030, dtype: datetime64[ns]
```

In [18]:
```
df_LOR['title'].value_counts()[:30]
```

Out[18]:
```
Debt consolidation            152472
Credit card refinancing        51487
Home improvement               15264
Other                          12930
Debt Consolidation             11608
Major purchase                  4769
Consolidation                   3852
debt consolidation              3547
Business                        2949
Debt Consolidation Loan         2864
Medical expenses                2742
Car financing                   2139
Credit Card Consolidation       1775
Vacation                        1717
Moving and relocation           1689
consolidation                   1595
Personal Loan                   1591
Consolidation Loan              1299
Home Improvement                1268
Home buying                     1183
Credit Card Refinance           1094
Credit Card Payoff              1052
Consolidate                      919
Personal                         858
Loan                             751
Credit Card Loan                 627
Freedom                          579
consolidate                      564
personal                         557
personal loan                    550
Name: title, dtype: int64
```

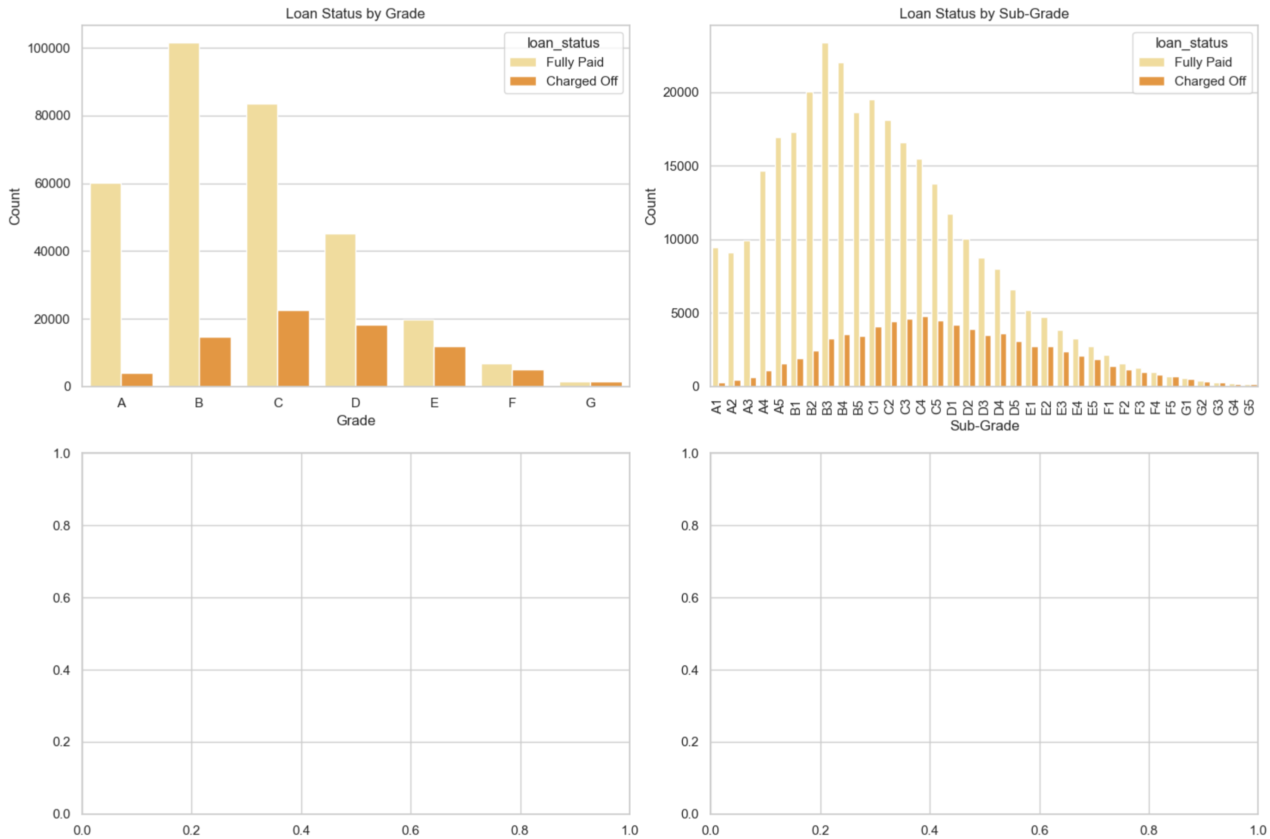
In [19]:
```
df_LOR['title']=df_LOR.title.str.lower()
```

In [20]:
```
df_LOR['title'].value_counts()[:30]
```

Out[20]:
```
debt consolidation           168108
credit card refinancing       51781
home improvement              17117
other                         12993
consolidation                  5583
major purchase                 4998
debt consolidation loan        3513
business                       3017
medical expenses               2820
credit card consolidation      2638
personal loan                  2460
car financing                  2160
credit card payoff             1904
consolidation loan             1887
vacation                       1866
credit card refinance          1832
moving and relocation          1693
consolidate                    1528
personal                       1465
home buying                    1196
loan                           1150
payoff                         1035
credit cards                   1030
freedom                         934
debt                            933
my loan                         897
credit card loan                879
credit card                     848
debt consolidation              840
debt free                       748
Name: title, dtype: int64
```

Data Visualization

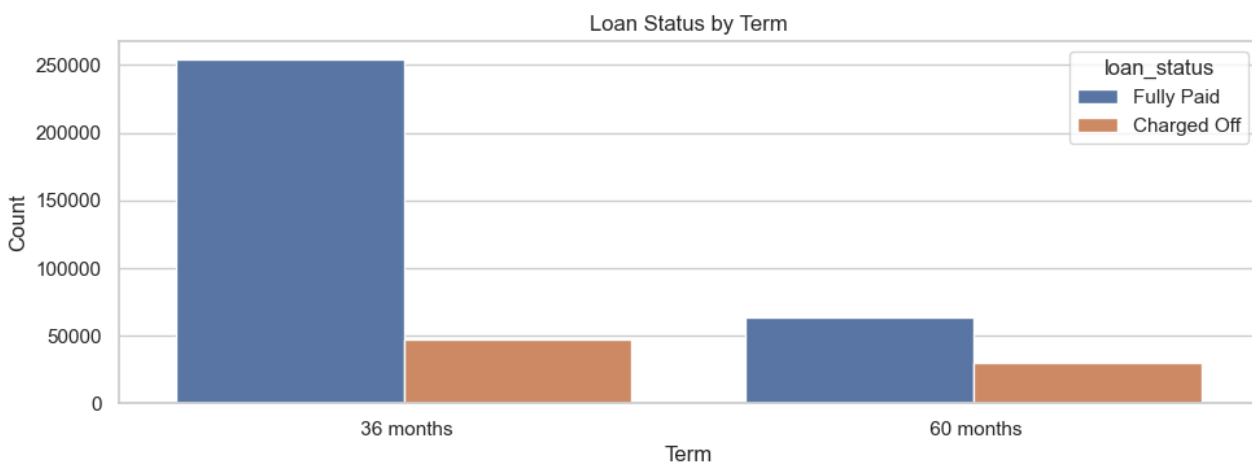
In [21]:
```
# Set Seaborn theme and palette for better visuals
sns.set_theme(style="whitegrid")
custom_palette = sns.color_palette("YlOrBr", n_colors=3)  # Choose a palette

# Set up the figure size
fig, axes = plt.subplots(2, 2, figsize=(15, 10))

# Plot 1: Countplot for 'grade'
grade = sorted(df_LOR['grade'].unique().tolist())
sns.countplot(
    x='grade',
    data=df_LOR,
    hue='loan_status',
    order=grade,
    ax=axes[0, 0],
    palette=custom_palette  # Apply custom palette
)
axes[0, 0].set_title("Loan Status by Grade")
axes[0, 0].set_xlabel("Grade")
axes[0, 0].set_ylabel("Count")

# Plot 2: Countplot for 'sub_grade'
sub_grade = sorted(df_LOR['sub_grade'].unique().tolist())
g = sns.countplot(
    x='sub_grade',
    data=df_LOR,
    hue='loan_status',
    order=sub_grade,
    ax=axes[0, 1],
    palette=custom_palette  # Apply custom palette
)
g.set_xticklabels(g.get_xticklabels(), rotation=90)
axes[0, 1].set_title("Loan Status by Sub-Grade")
axes[0, 1].set_xlabel("Sub-Grade")
axes[0, 1].set_ylabel("Count")

# Adjust layout for better spacing
plt.tight_layout()
plt.show()
```

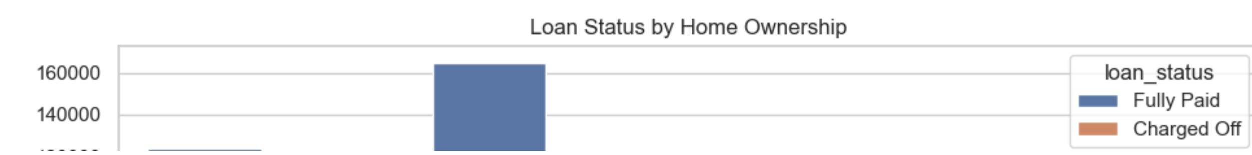


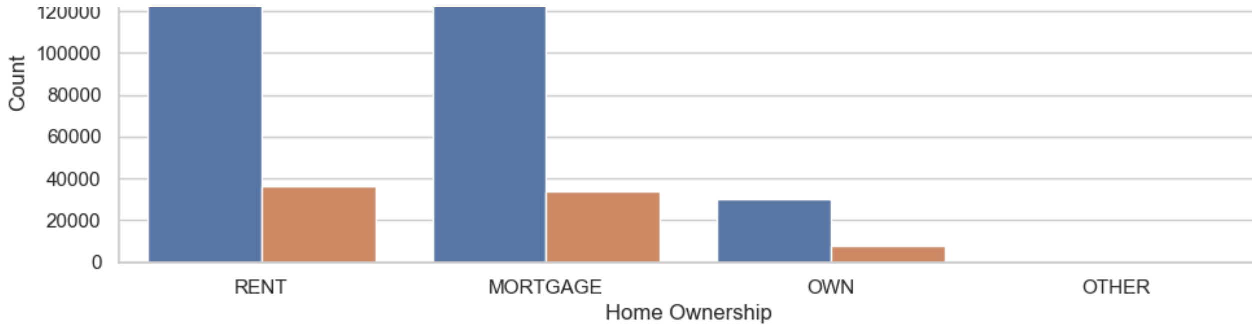
Observation: The majority of individuals who have fully paid their loans fall under grade 'B' with a subgrade of 'B3'.

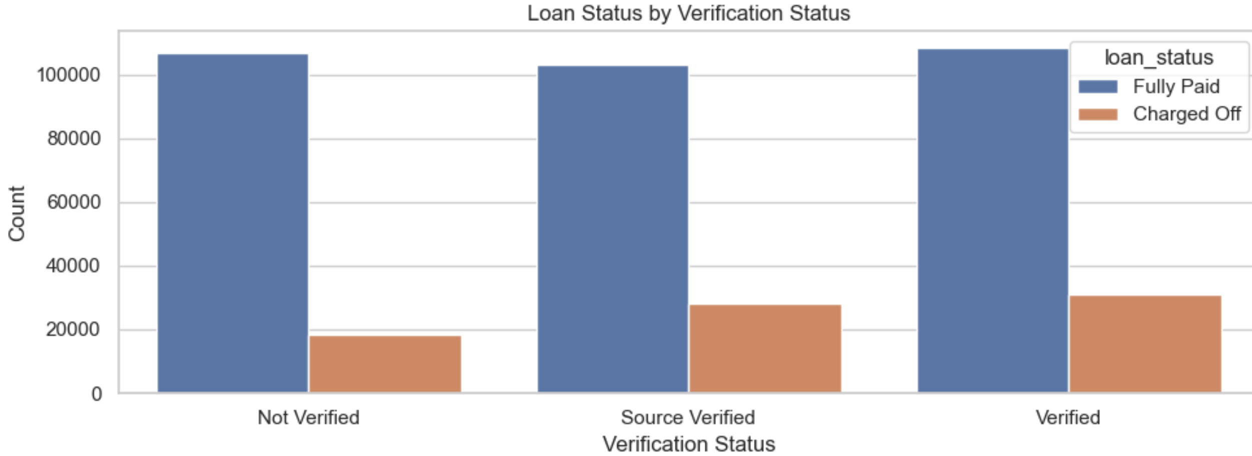
This suggests that people with grade 'B' and subgrade 'B3' are more likely to fully repay their loans.

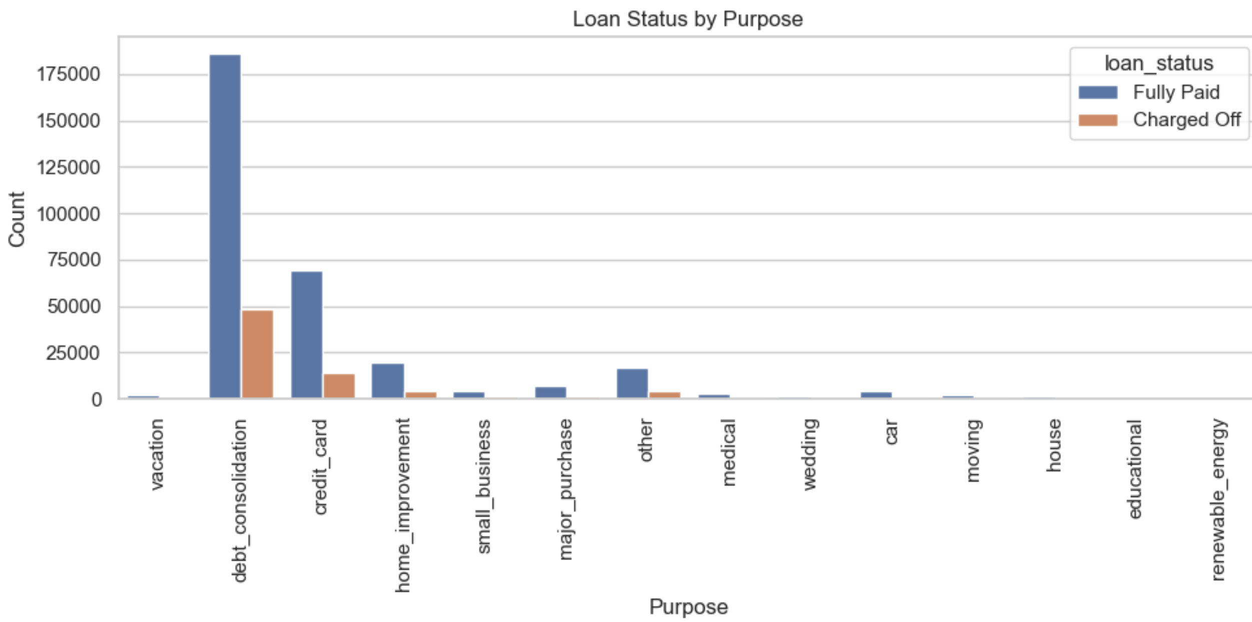
In [22]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Set up the figure and axes for 4 subplots
fig, axes = plt.subplots(4, 1, figsize=(10, 16))  # 4 rows, 1 column

# Plot 1: Term vs Loan Status
sns.countplot(x='term', data=df_LOR, hue='loan_status', ax=axes[0])
axes[0].set_title("Loan Status by Term")
axes[0].set_xlabel("Term")
axes[0].set_ylabel("Count")

# Plot 2: Home Ownership vs Loan Status
sns.countplot(x='home_ownership', data=df_LOR, hue='loan_status', ax=axes[1]
axes[1].set_title("Loan Status by Home Ownership")
axes[1].set_xlabel("Home Ownership")
axes[1].set_ylabel("Count")

# Plot 3: Verification Status vs Loan Status
sns.countplot(x='verification_status', data=df_LOR, hue='loan_status', ax=ax
axes[2].set_title("Loan Status by Verification Status")
axes[2].set_xlabel("Verification Status")
axes[2].set_ylabel("Count")

# Plot 4: Purpose vs Loan Status
g = sns.countplot(x='purpose', data=df_LOR, hue='loan_status', ax=axes[3])
axes[3].set_title("Loan Status by Purpose")
axes[3].set_xlabel("Purpose")
axes[3].set_ylabel("Count")
g.set_xticklabels(g.get_xticklabels(), rotation=90)

# Adjust layout for better spacing
plt.tight_layout()
plt.show()
```







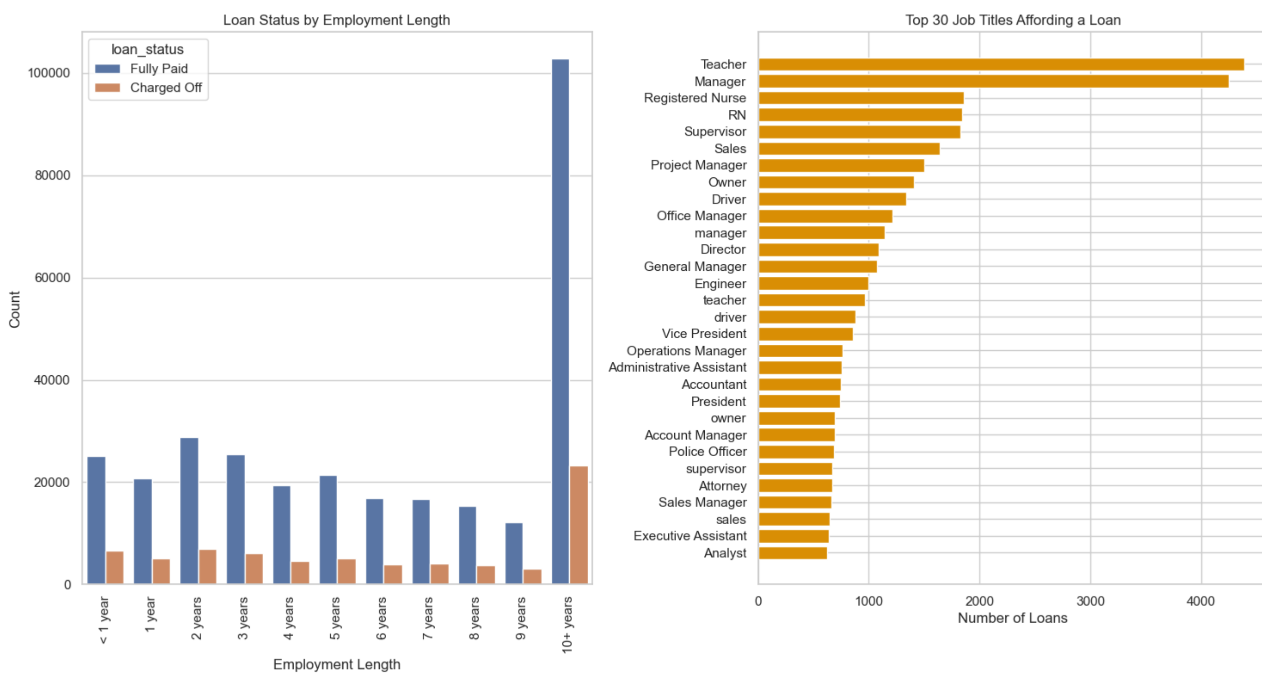




Observation: Loans with a '36 months' term have the highest number of 'Fully Paid' cases in the loan_status category. Similarly, 'Mortgage' leads in the 'Fully Paid' category under home ownership, while for purpose, the majority of fully paid loans fall under the 'debt_consolidation' category.

emp_length vs loan_status

In [23]:
```
# Set up the figure and axes for two subplots
fig, axes = plt.subplots(1, 2, figsize=(15, 8))  # 1 row, 2 columns

# Plot 1: Employment Length vs Loan Status
order = ['< 1 year', '1 year', '2 years', '3 years', '4 years', '5 years',
         '6 years', '7 years', '8 years', '9 years', '10+ years']
sns.countplot(
    x='emp_length',
    data=df_LOR,
    hue='loan_status',
    order=order,
    ax=axes[0]
)
axes[0].set_title("Loan Status by Employment Length")
axes[0].set_xlabel("Employment Length")
axes[0].set_ylabel("Count")
axes[0].set_xticklabels(axes[0].get_xticklabels(), rotation=90)

# Plot 2: Top 30 Job Titles Affording Loans
top_30_jobs = df_LOR['emp_title'].value_counts()[:30]
axes[1].barh(top_30_jobs.index, top_30_jobs.values, color='#D98E04')
axes[1].set_title("Top 30 Job Titles Affording a Loan")
axes[1].set_xlabel("Number of Loans")
axes[1].invert_yaxis()  # Ensure the top job appears at the top

# Adjust layout for better spacing
plt.tight_layout()
plt.show()
```



Observation: It is clear that long-term loans, such as those spanning 10+ years, are predominantly paid fully by the customers.

Additionally, "Teacher" and "Manager" are the job titles that have secured the highest number of loans.

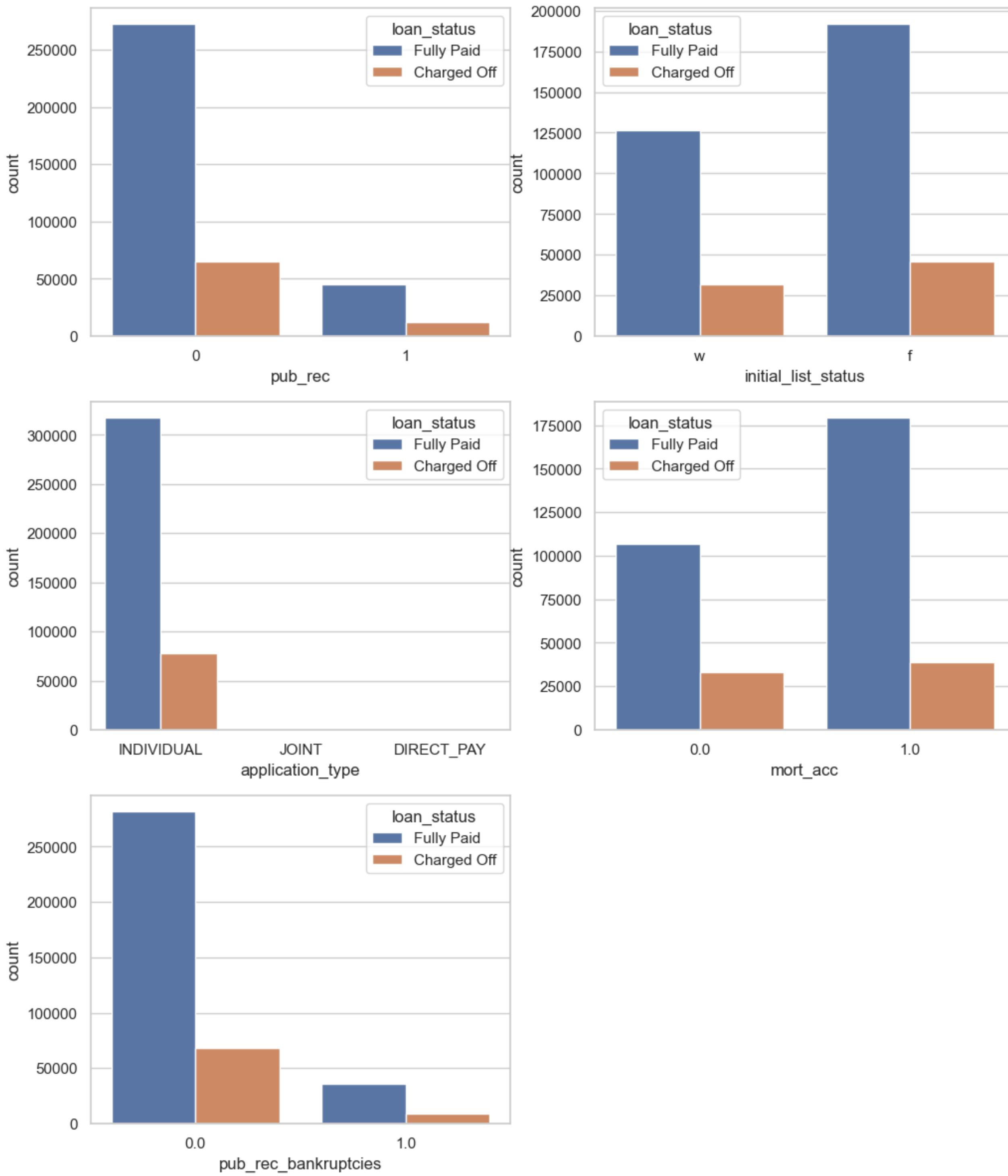
Feature Engineering

In [24]:
```
def pub_rec(number):
    if number == 0.0:
        return 0
    else:
        return 1

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


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

In [25]:
```
df_LOR['pub_rec']=df_LOR.pub_rec.apply(pub_rec)
df_LOR['mort_acc']=df_LOR.mort_acc.apply(mort_acc)
df_LOR['pub_rec_bankruptcies']=df_LOR.pub_rec_bankruptcies.apply(pub_rec_ban
```

In [26]:
```
plt.figure(figsize=(12,30))

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

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

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

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

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

Out[26]:
```
<Axes: xlabel='pub_rec_bankruptcies', ylabel='count'>
```



Mapping of Target Variable

In [27]:
```
df_LOR['loan_status']=df_LOR.loan_status.map({'Fully Paid':0, 'Charged Off':
```

Mean Target Imputation

In [28]:
```
df_LOR.groupby(by='total_acc').mean()
```

Out[28]:

	loan_amnt	int_rate	annual_inc	loan_status	dti	open_acc	p
total_acc							
2.0	6672.222222	15.801111	64277.777778	0.222222	2.279444	1.611111	0.(
3.0	6042.966361	15.615566	41270.753884	0.220183	6.502813	2.611621	0.(
4.0	7587.399031	15.069491	42426.565969	0.214055	8.411963	3.324717	0.
5.0	7845.734714	14.917564	44394.098003	0.203156	10.118328	3.921598	0.(
6.0	8529.019843	14.651752	48470.001156	0.215874	11.222542	4.511119	0.(
...	...	...	...	...	...	...	
124.0	23200.000000	17.860000	66000.000000	1.000000	14.040000	43.000000	0.(
129.0	25000.000000	7.890000	200000.000000	0.000000	8.900000	48.000000	0.(
135.0	24000.000000	15.410000	82000.000000	0.000000	33.850000	57.000000	0.(
150.0	35000.000000	8.670000	189000.000000	0.000000	6.630000	40.000000	0.(
151.0	35000.000000	13.990000	160000.000000	1.000000	12.650000	26.000000	0.(

118 rows × 11 columns

In [29]:
```
# saving mean of mort_acc according to total_acc_avg
total_acc_avg=df_LOR.groupby(by='total_acc').mean().mort_acc
```

In [30]:
```
def fill_mort_acc(total_acc,mort_acc):
    if np.isnan(mort_acc):
        return total_acc_avg[total_acc].round()
    else:
        return mort_acc
```

In [31]:
```
df_LOR['mort_acc']=df_LOR.apply(lambda x: fill_mort_acc(x['total_acc'],x['mo
```

In [32]:
```
df_LOR.isnull().sum()/len(df_LOR)*100
```

Out[32]:
```
loan_amnt               0.000000
term                    0.000000
int_rate                0.000000
grade                   0.000000
sub_grade               0.000000
emp_title               5.789208
emp_length              4.621115
home_ownership          0.000000
annual_inc              0.000000
verification_status     0.000000
issue_d                 0.000000
loan_status             0.000000
purpose                 0.000000
title                   0.443148
dti                     0.000000
earliest_cr_line        0.000000
open_acc                0.000000
pub_rec                 0.000000
revol_bal               0.000000
revol_util              0.069692
total_acc               0.000000
initial_list_status     0.000000
application_type        0.000000
mort_acc                0.000000
pub_rec_bankruptcies    0.135091
address                 0.000000
dtype: float64
```

In [33]:
```
# Dropping rows with null values
df_LOR.dropna(inplace=True)
```

In [34]:
```
df_LOR
```

Out[34]:

	loan_amnt	term	int_rate	grade	sub_grade	emp_title	emp_length	home_o
0	10000.0	36 months	11.44	B	B4	Marketing	10+ years	
1	8000.0	36 months	11.99	B	B5	Credit analyst	4 years	M(
2	15600.0	36 months	10.49	B	B3	Statistician	< 1 year	
3	7200.0	36 months	6.49	A	A2	Client Advocate	6 years	
4	24375.0	60 months	17.27	C	C5	Destiny Management Inc.	9 years	M(
...	...	...	...	...	...	...	...	
396025	10000.0	60 months	10.99	B	B4	licensed bankere	2 years	
396026	21000.0	36 months	12.29	C	C1	Agent	5 years	M(
396027	5000.0	36 months	9.99	B	B1	City Carrier	10+ years	
396028	21000.0	60 months	15.31	C	C2	Gracon Services, Inc	10+ years	M(
396029	2000.0	36 months	13.61	C	C2	Internal Revenue Service	10+ years	

370622 rows × 26 columns

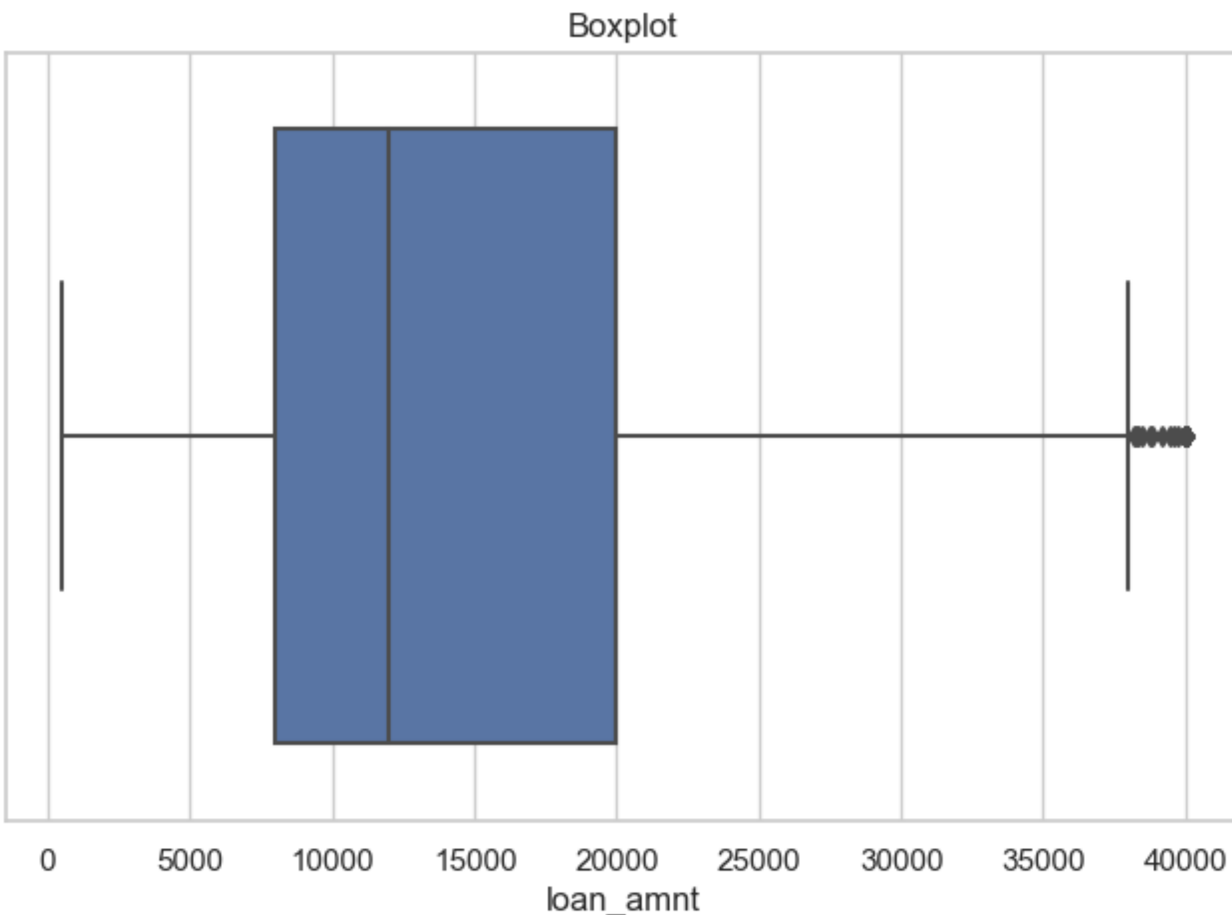
Detection of Outlier & Treatment

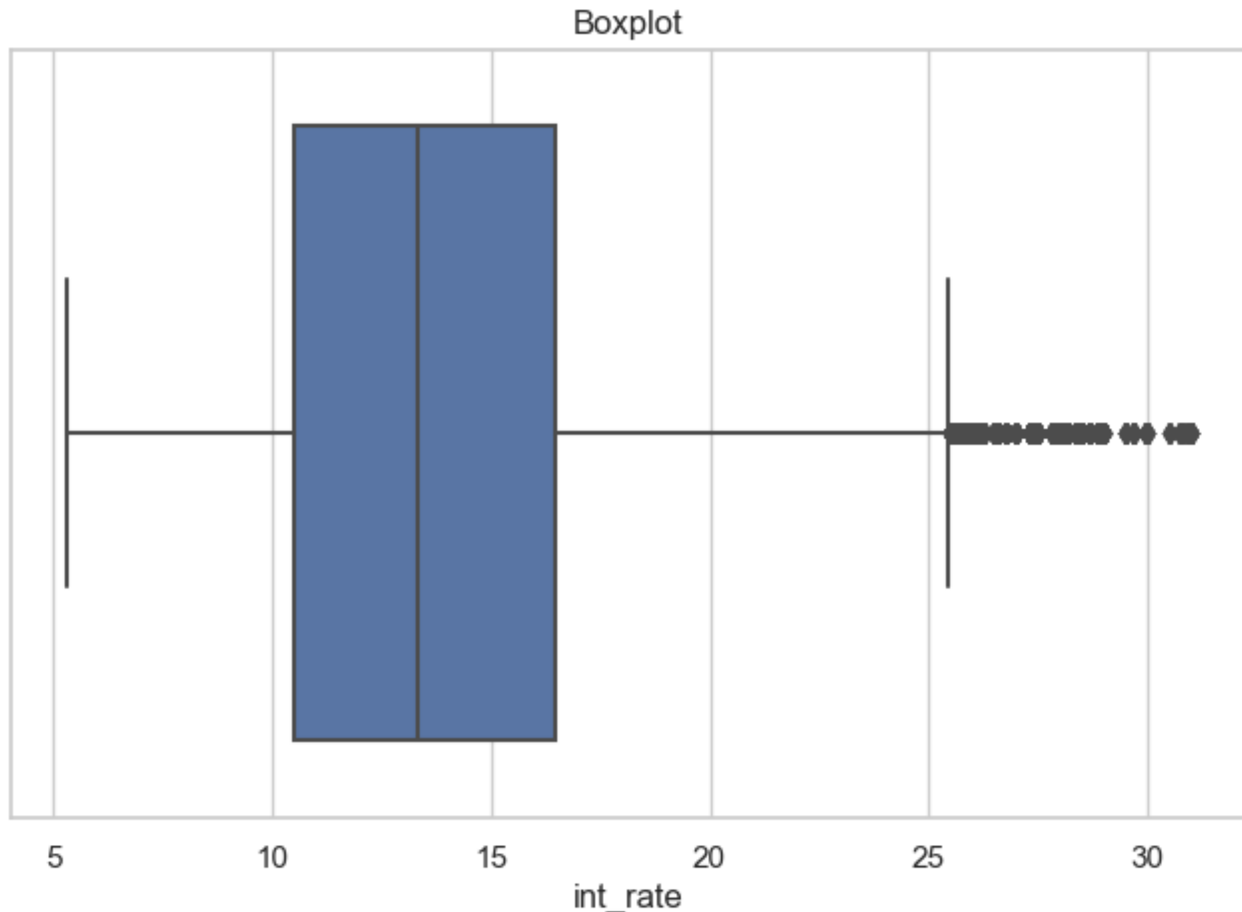
In [35]:

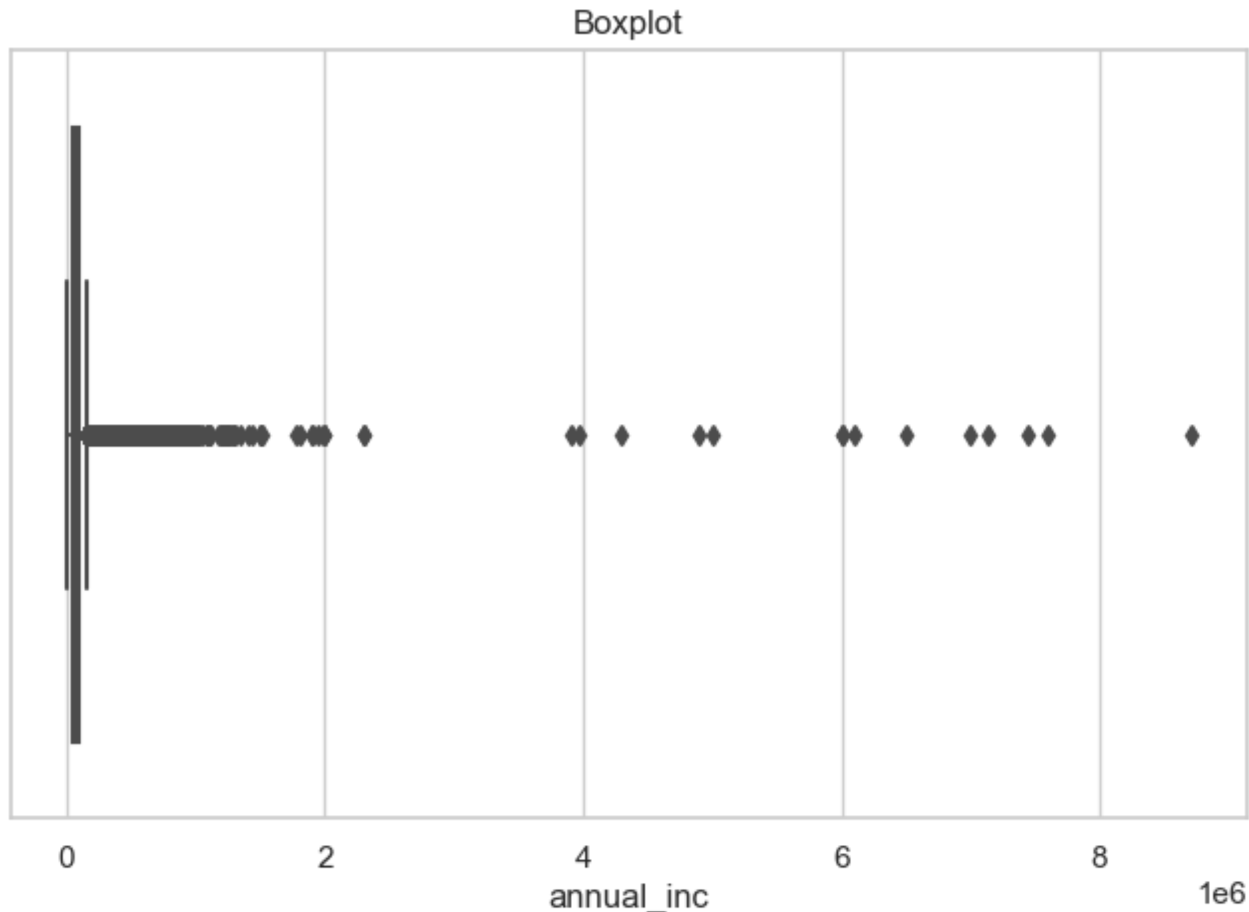
```
num_data=df_LOR.select_dtypes(include='number')
num_cols=num_data.columns
len(num_cols)
```

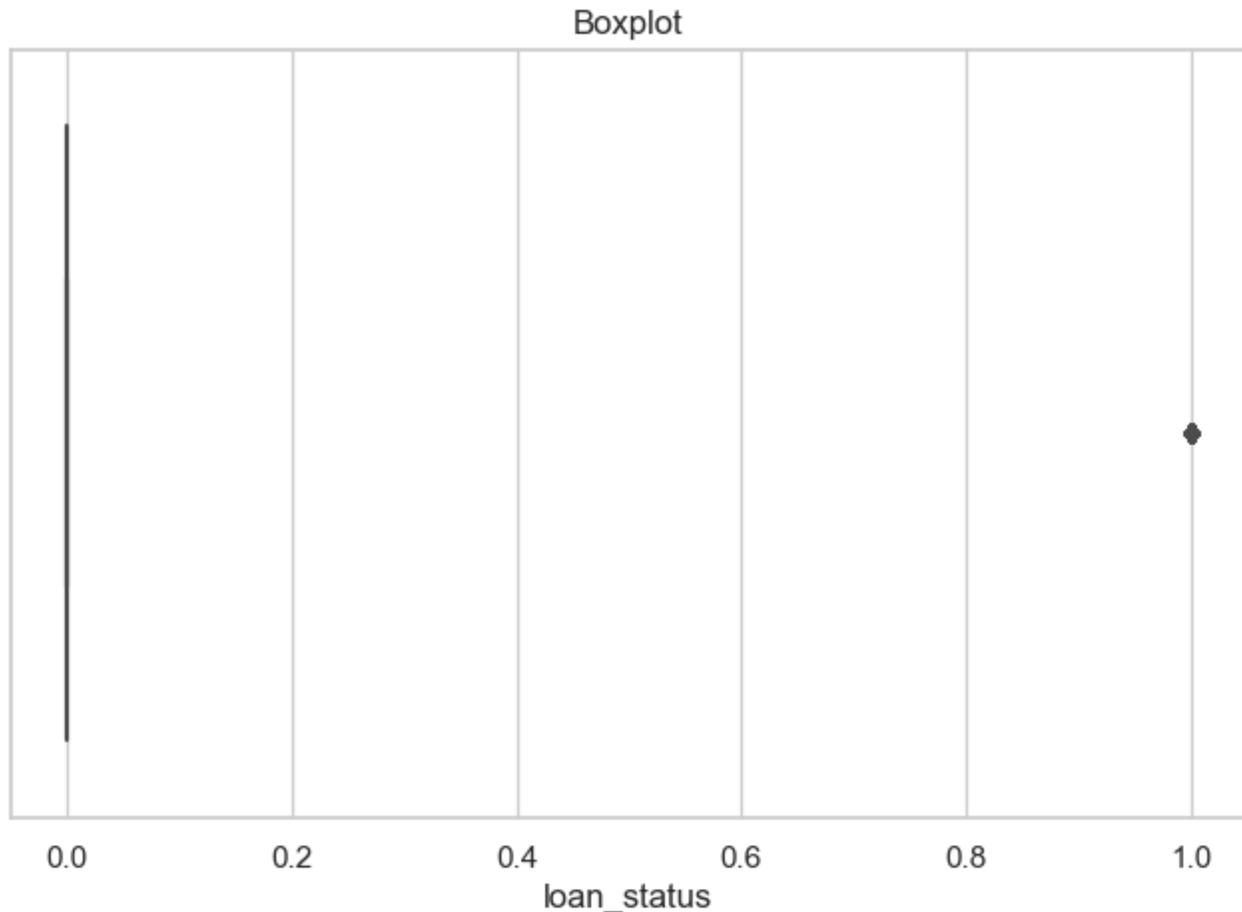
Out[35]: 12

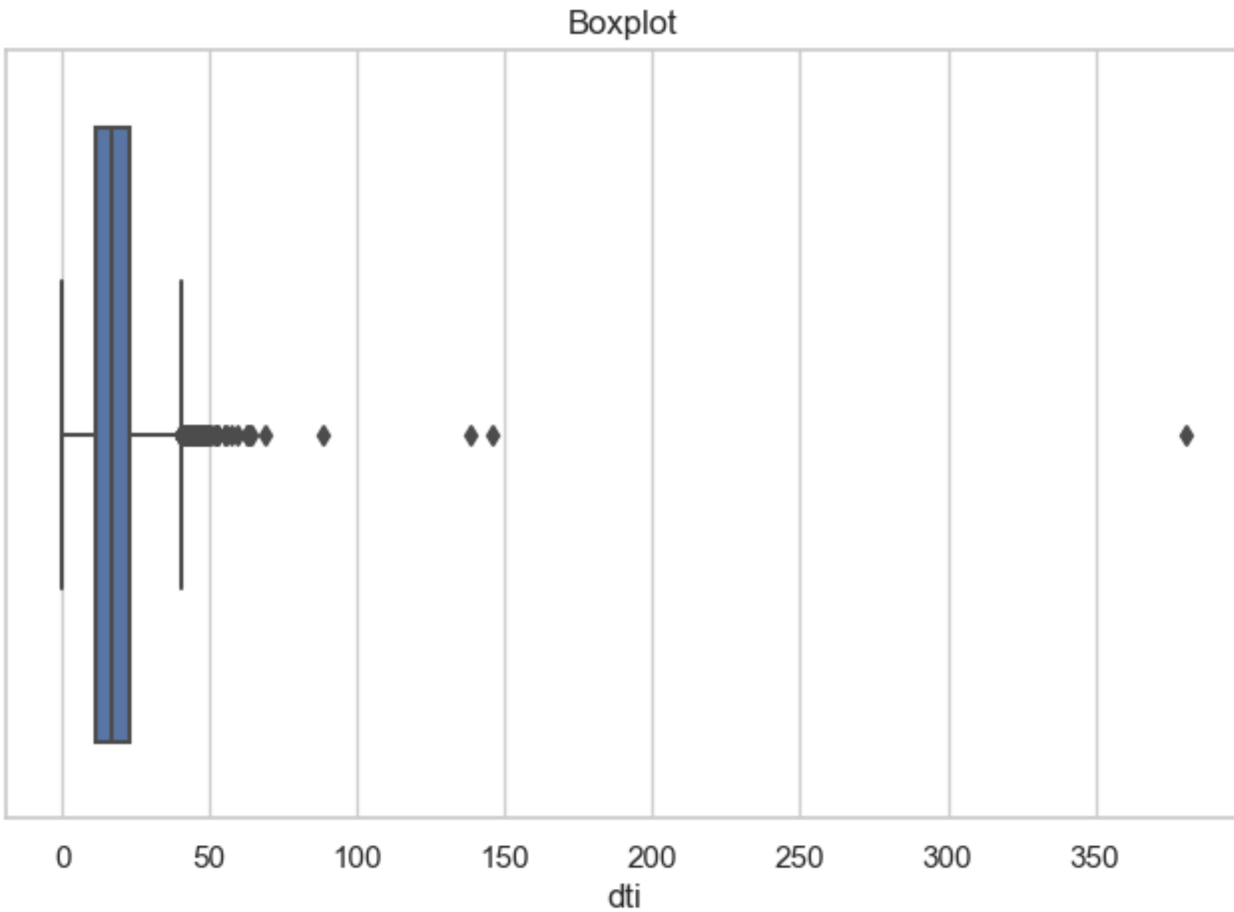
In [36]:
```
def box_plot(col):
    plt.figure(figsize=(8,5))
    sns.boxplot(x=df_LOR[col])
    plt.title('Boxplot')
    plt.show()

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

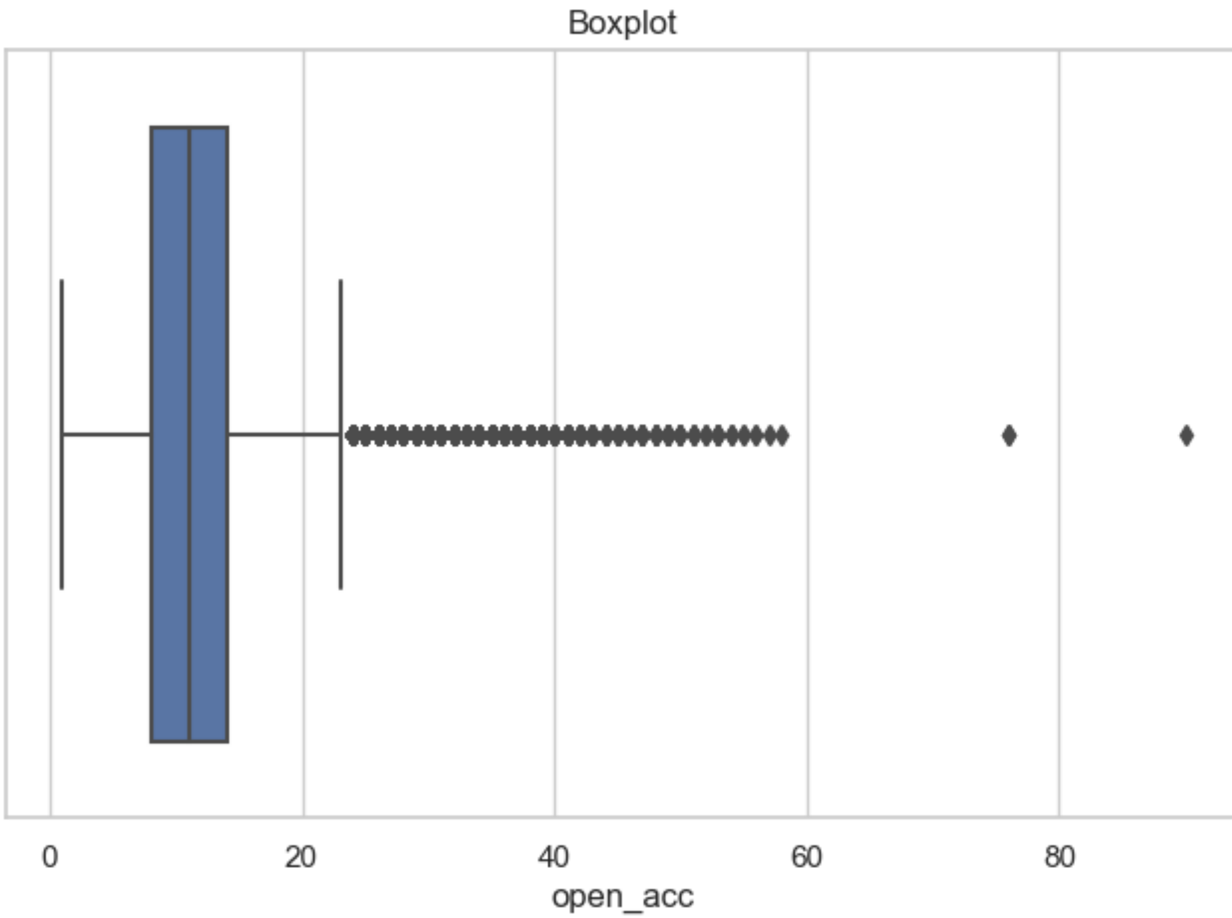


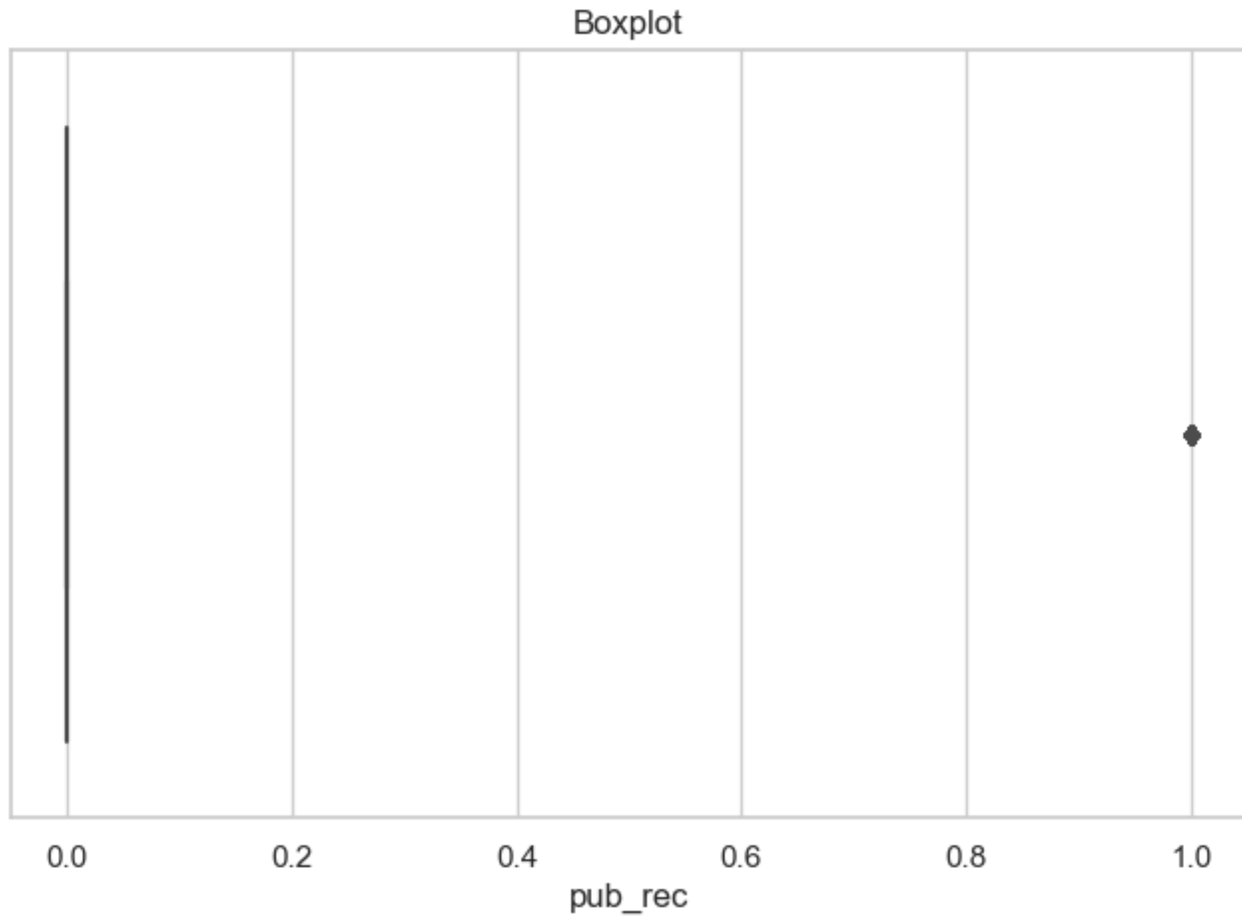

Boxplot
5
10
15
20
25
30
int_rate

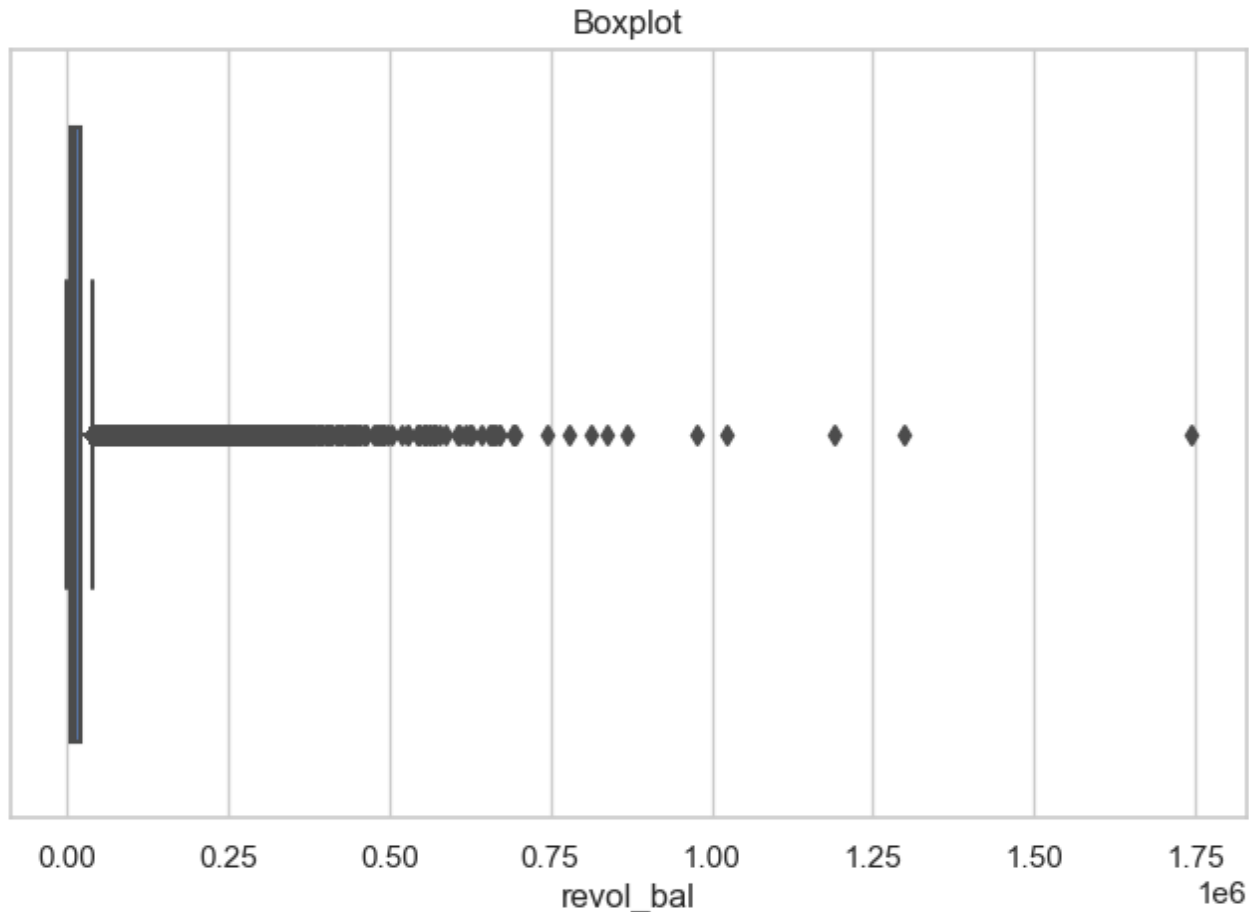

Boxplot
0
2
4
6
8
1e6
annual_inc

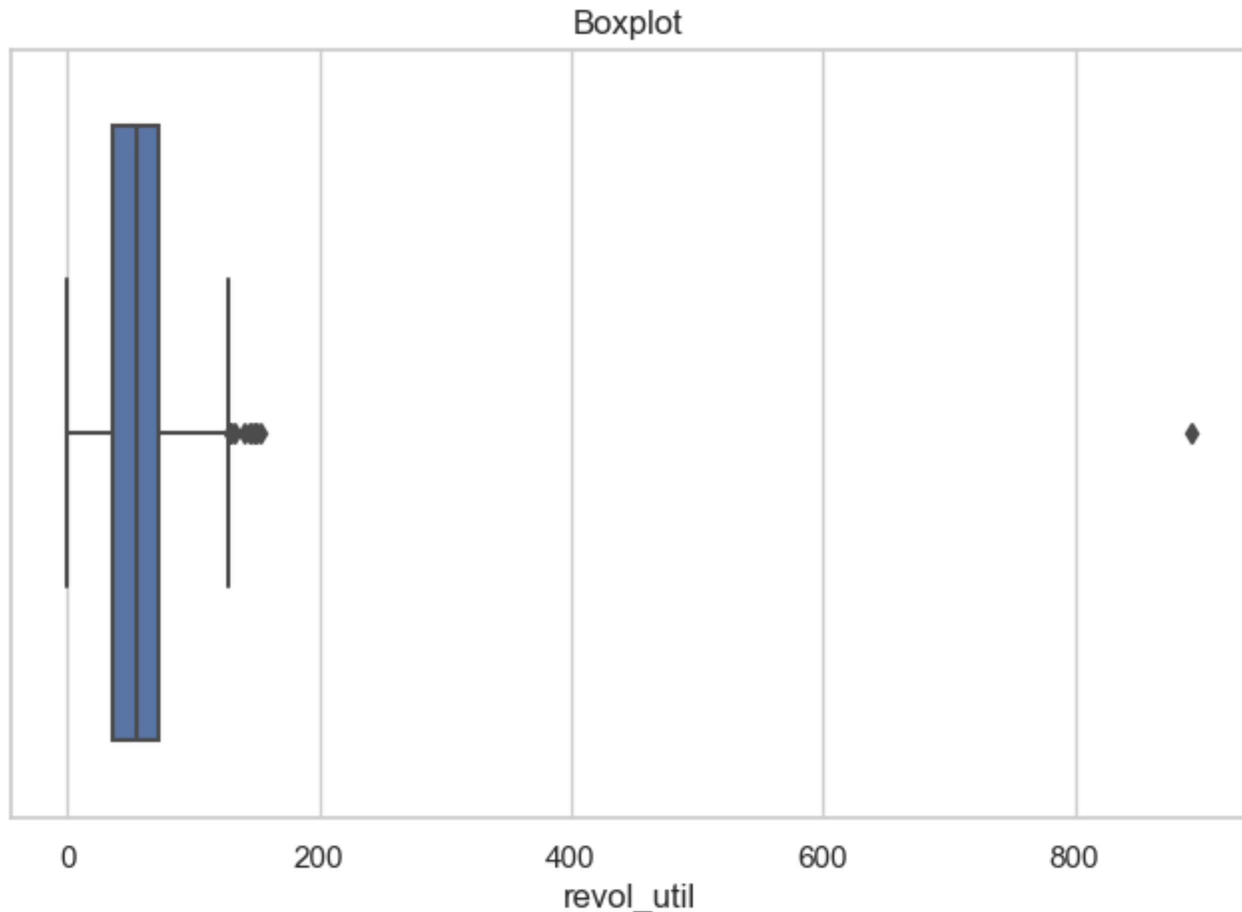

Boxplot
0.0
0.2
0.4
0.6
0.8
1.0
loan_status

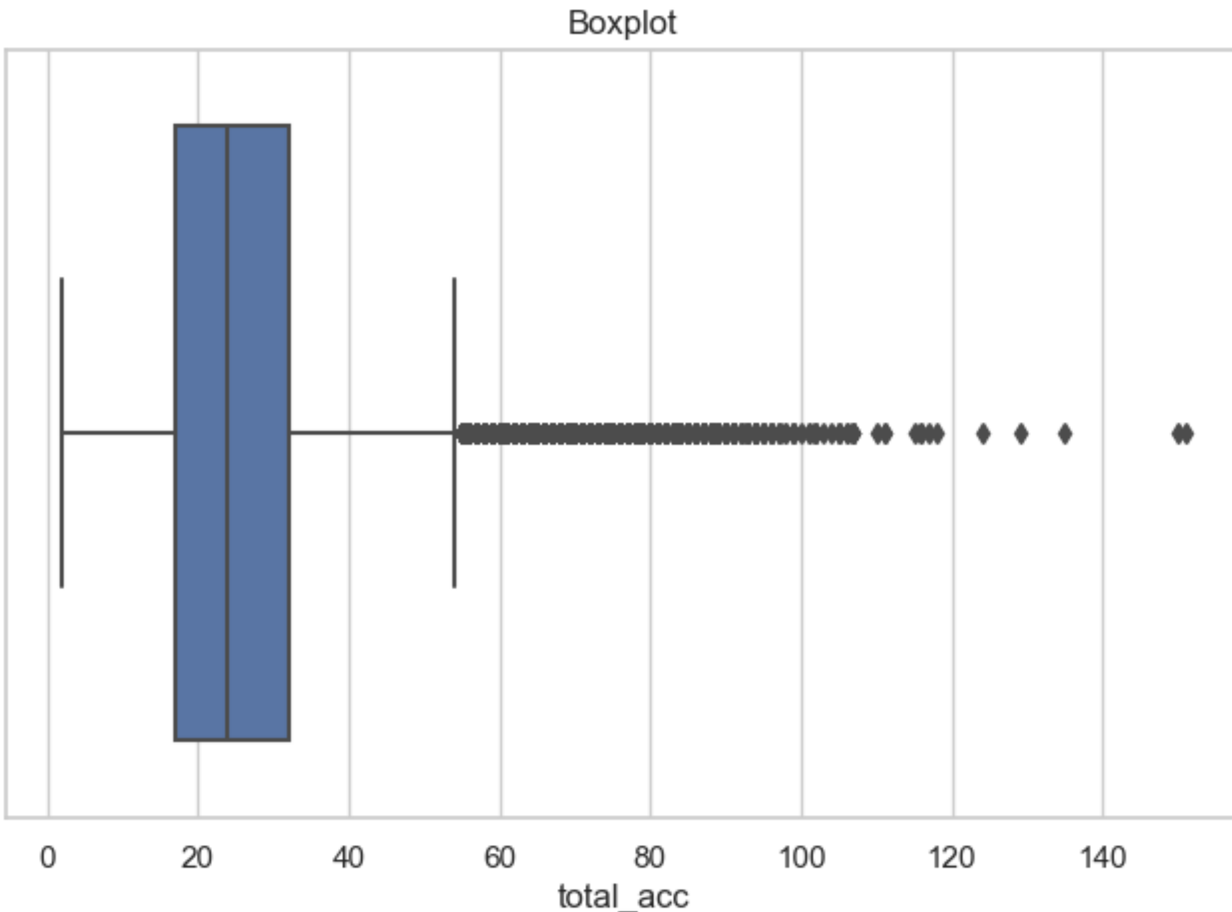

Boxplot
0
50
100
150
200
250
300
350
dti

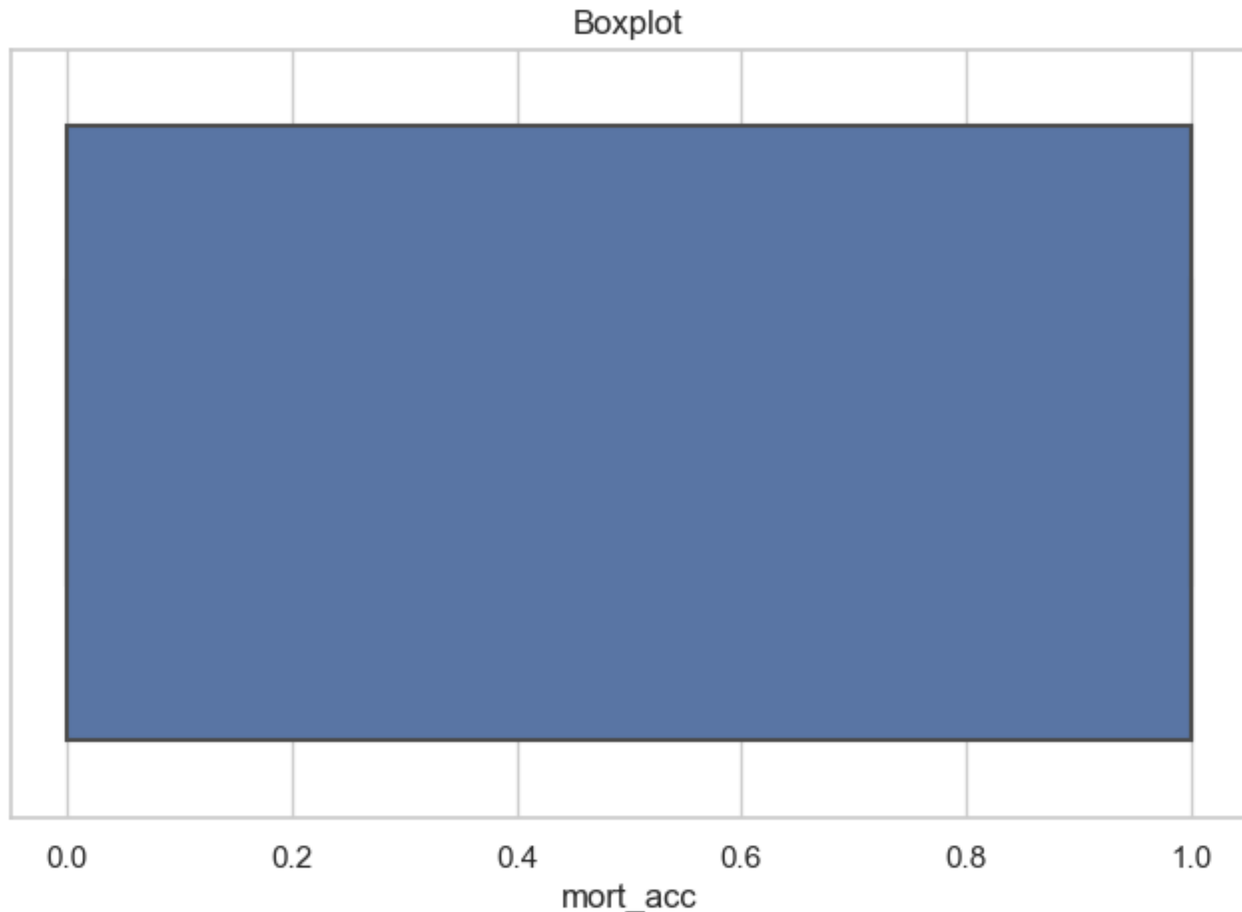

Boxplot
0
20
40
60
80
open_acc

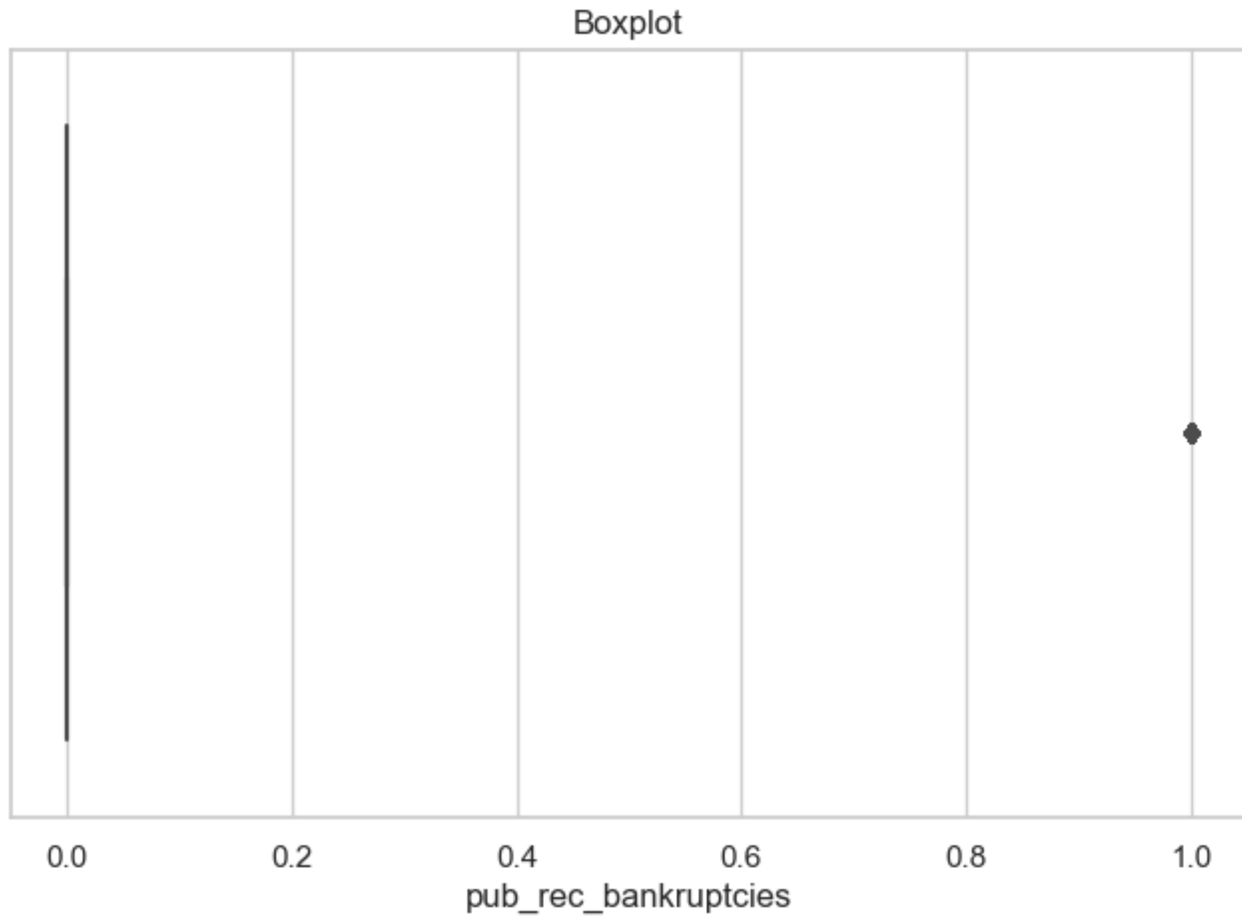

Boxplot
0.0
0.2
0.4
0.6
0.8
1.0
pub_rec


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


Boxplot
0
200
400
600
800
revol_util


Boxplot
0
20
40
60
80
100
120
140
total_acc


Boxplot
0.0
0.2
0.4
0.6
0.8
1.0
mort_acc



In [37]:
```
for col in num_cols:
    mean=df_LOR[col].mean()
    std=df_LOR[col].std()

    upper_limit=mean+3*std
    lower_limit=mean-3*std

    df_LOR=df_LOR[(df_LOR[col]<upper_limit) & (df_LOR[col]>lower_limit)]

df_LOR.shape
```

Out[37]:
```
(354519, 26)
```

Data Preprocessing

Mapping term column

In [38]:
```
df_LOR.term.unique()
```

Out[38]:
```
array([' 36 months', ' 60 months'], dtype=object)
```

In [39]:
```
term_values={' 36 months': 36, ' 60 months':60}
df_LOR['term'] = df_LOR.term.map(term_values)
```

In [40]:
```
# Initial List Status
df_LOR['initial_list_status'].unique()
```

Out[40]:
```
array(['w', 'f'], dtype=object)
```

In [41]:
```
list_status = {'w': 0, 'f': 1}
df_LOR['initial_list_status'] = df_LOR.initial_list_status.map(list_status)
```

In [42]:
```
# Let's fetch ZIP from address and then drop the remaining details:
df_LOR['zip_code'] = df_LOR.address.apply(lambda x: x[-5:])
```

In [43]:
```
df_LOR['zip_code'].value_counts(normalize=True)*100
```

Out[43]:
```
70466    14.382022
30723    14.277373
22690    14.268347
48052    14.127028
00813    11.610097
29597    11.537322
05113    11.516731
93700     2.774746
11650     2.772771
86630     2.733563
Name: zip_code, dtype: float64
```

In [44]:
```
# Dropping some variables which we can let go for now
df_LOR.drop(columns=['issue_d', 'emp_title', 'title', 'sub_grade',
                   'address', 'earliest_cr_line', 'emp_length'],
                   axis=1, inplace=True)
```

In [45]:
```
df_LOR
```

Out[45]:

	loan_amnt	term	int_rate	grade	home_ownership	annual_inc	verification_status
0	10000.0	36	11.44	B	RENT	117000.0	Not Verified
1	8000.0	36	11.99	B	MORTGAGE	65000.0	Not Verified
2	15600.0	36	10.49	B	RENT	43057.0	Source Verified
3	7200.0	36	6.49	A	RENT	54000.0	Not Verified
4	24375.0	60	17.27	C	MORTGAGE	55000.0	Verified
...	...	...	...	...	...	...	...
396025	10000.0	60	10.99	B	RENT	40000.0	Source Verified
396026	21000.0	36	12.29	C	MORTGAGE	110000.0	Source Verified
396027	5000.0	36	9.99	B	RENT	56500.0	Verified
396028	21000.0	60	15.31	C	MORTGAGE	64000.0	Verified
396029	2000.0	36	13.61	C	RENT	42996.0	Verified

354519 rows × 20 columns

One-hot Encoding

In [46]:
```
dummies=['purpose', 'zip_code', 'grade', 'verification_status', 'application
df_LOR=pd.get_dummies(df_LOR,columns=dummies,drop_first=True)
```

In [47]:
```
pd.set_option('display.max_columns',None)
pd.set_option('display.max_rows',None)

df_LOR.head()
```

Out[47]:

	loan_amnt	term	int_rate	annual_inc	loan_status	dti	open_acc	pub_rec	revol_bal
0	10000.0	36	11.44	117000.0	0	26.24	16.0	0	36369.0
1	8000.0	36	11.99	65000.0	0	22.05	17.0	0	20131.0
2	15600.0	36	10.49	43057.0	0	12.79	13.0	0	11987.0
3	7200.0	36	6.49	54000.0	0	2.60	6.0	0	5472.0
4	24375.0	60	17.27	55000.0	1	33.95	13.0	0	24584.0

In [48]:
```
df_LOR.shape
```

Out[48]:
```
(354519, 49)
```

Data preparation for MOdelling

In [49]:
```
X=df_LOR.drop('loan_status',axis=1)
y=df_LOR['loan_status']
```

In [50]:
```
X_train, X_test, y_train, y_test =train_test_split(X,y,test_size=0.30,strati
```

In [51]:
```
print(X_train.shape)
print(X_test.shape)
```

```
(248163, 48)
(106356, 48)
```

For each value in a feature, the MinMaxScaler subtracts the minimum value of the feature and divides the result by the feature's range, where the range is the difference between the original maximum and minimum values.

In [52]:
```
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Logistic Regression

In [53]:
```
logreg=LogisticRegression(max_iter=1000)
logreg.fit(X_train,y_train)
```

Out[53]:
```
▾ LogisticRegression
LogisticRegression(max_iter=1000)
```

In [54]:
```
y_pred = logreg.predict(X_test)
print('Accuracy of Logistic Regression Classifier on test set: {:.3f}'.forma
```

```
Accuracy of Logistic Regression Classifier on test set: 0.890
```

Confusion Matrix

In [55]:
```
confusion_matrix=confusion_matrix(y_test,y_pred)
print(confusion_matrix)
```

```
[[85364   524]
 [11132  9336]]
```

Classificatiion Report

In [56]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.88      0.99      0.94     85888
           1       0.95      0.46      0.62     20468

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

ROC Curve

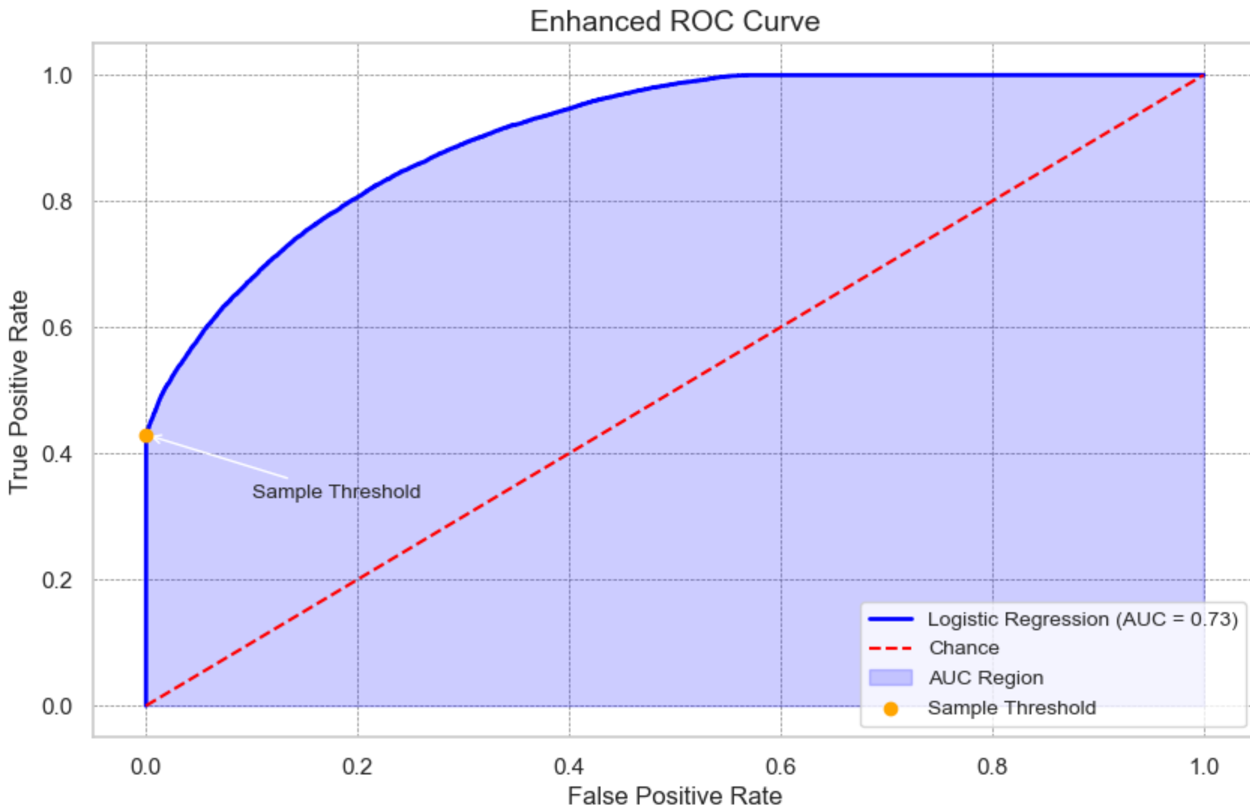
ROC Curve -

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

True Positive Rate False Positive Rate

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

In [59]:
```
# ROC Curve
plt.figure(figsize=(10, 6))

# Plot ROC Curve
plt.plot(fpr, tpr, label='Logistic Regression (AUC = %.2f)' % logit_roc_auc,

# Diagonal line for random guessing
plt.plot([0, 1], [0, 1], linestyle='--', color='red', label='Chance')

# Shade the AUC region
plt.fill_between(fpr, tpr, alpha=0.2, color='blue', label='AUC Region')

# Annotate a specific threshold
plt.scatter(fpr[50], tpr[50], color='orange', label='Sample Threshold', zord
plt.annotate('Sample Threshold', xy=(fpr[50], tpr[50]),
             xytext=(fpr[50] + 0.1, tpr[50] - 0.1),
             arrowprops=dict(facecolor='black', arrowstyle='->'), fontsize=1

# Add labels and title
plt.xlabel('False Positive Rate', fontsize=12)
plt.ylabel('True Positive Rate', fontsize=12)
plt.title('Enhanced ROC Curve', fontsize=14)

# Add grid and legend
plt.grid(color='gray', linestyle='--', linewidth=0.5)
plt.legend(loc='lower right', fontsize=10)

# Show the plot
plt.show()
```



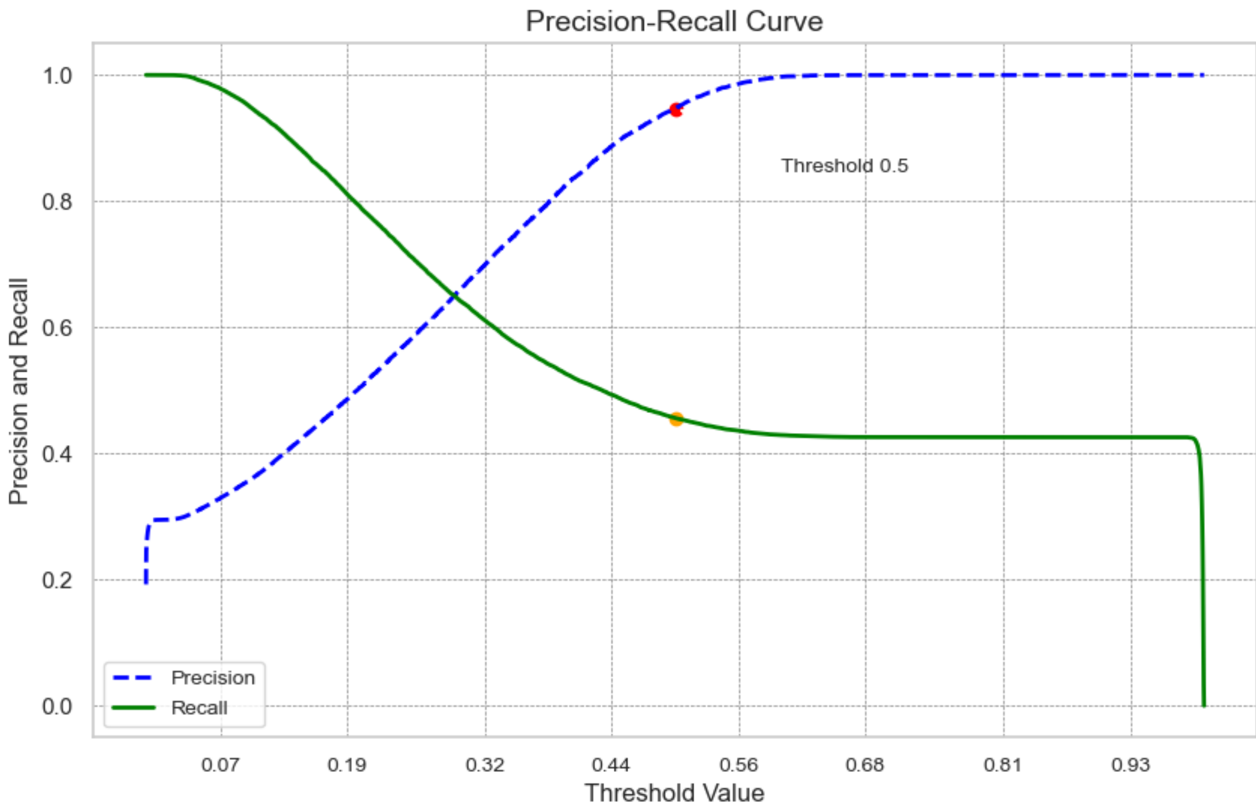
AUC represents the "Area Under the ROC Curve," measuring the total two-dimensional area beneath the entire ROC curve, spanning from (0,0) to (1,1). It offers a comprehensive measure of a model's performance across all possible classification thresholds.

Precision & Recall

Plots the Precision-Recall Curve for the given test labels and predicted probabilities.

Parameters: y_test: True binary labels for the test data. pred_proba_c1: Predicted probabilities for the positive class (class 1).

In [62]:
```
def precision_recall_curve_plot(y_test, pred_proba_c1):

    # Compute precision, recall, and thresholds
    precisions, recalls, thresholds = precision_recall_curve(y_test, pred_pr

    # Plot precision and recall values against thresholds
    plt.figure(figsize=(10, 6))  # Set figure size
    plt.plot(thresholds, precisions[:-1], linestyle='--', label='Precision',
    plt.plot(thresholds, recalls[:-1], label='Recall', color='green', linewi

    # Format the x-axis with rounded tick intervals
    start, end = plt.xlim()
    plt.xticks(np.round(np.linspace(start, end, 10), 2), fontsize=10)

    # Add labels, title, and legend
    plt.xlabel('Threshold Value', fontsize=12)
    plt.ylabel('Precision and Recall', fontsize=12)
    plt.title('Precision-Recall Curve', fontsize=14)
    plt.legend(loc='best', fontsize=10)
    plt.grid(color='gray', linestyle='--', linewidth=0.5)

    # Optional: Annotate a specific threshold point (e.g., 0.5)
    optimal_threshold = 0.5
    closest_index = np.argmin(np.abs(thresholds - optimal_threshold))
    plt.scatter(thresholds[closest_index], precisions[closest_index], color=
    plt.scatter(thresholds[closest_index], recalls[closest_index], color='or
    plt.annotate('Threshold 0.5',
                 xy=(thresholds[closest_index], precisions[closest_index]),
                 xytext=(thresholds[closest_index] + 0.1, precisions[closest
                 arrowprops=dict(facecolor='black', arrowstyle='->'), fontsi

    # Display the plot
    plt.show()

# Call the function with your data
precision_recall_curve_plot(y_test, logreg.predict_proba(X_test)[:, 1])
```



Observation : This plot will clearly display precision and recall values for different thresholds. A red marker and annotation will highlight the threshold = 0.5, allowing you to focus on the default classification boundary.

Multicollinearity Check

Multicollinearity happens when two or more independent variables in a regression model are highly correlated. This makes it hard to separate their individual impact on the dependent variable.

One way to detect multicollinearity is by using the Variance Inflation Factor (VIF). In this method, each independent variable is regressed against all the other independent variables. The VIF score shows how much a variable is explained by the others.

Calculate the Variance Inflation Factor (VIF) for each feature in the given DataFrame.

Parameters: dataframe (pd.DataFrame): The input DataFrame containing features.

Returns: pd.DataFrame: A DataFrame with features and their corresponding VIF values, sorted in descending order.

In [82]:
```
def calculate_vif(dataframe):

    # Initialize a DataFrame to store VIF values
    vif_data = pd.DataFrame()
    vif_data['Feature'] = X.columns

    # Calculate VIF for each feature
    vif_data['VIF'] = [variance_inflation_factor(X.values, i) for i in range

    # Round off VIF values to 2 decimal places
    vif_data['VIF'] = vif_data['VIF'].round(2)

    # Sort by VIF in descending order
    vif_data = vif_data.sort_values(by='VIF', ascending=False).reset_index(d

    return vif_data

# Call the function and display the top 5 features with the highest VIF
vif_results = calculate_vif(X)
vif_results.head(5)
```

Out[82]:

	Feature	VIF
0	term	23.35
1	purpose_debt_consolidation	22.35
2	open_acc	13.64
3	total_acc	12.69
4	revol_util	9.06

In [87]:
```
if 'term' in X.columns:
    X.drop(columns=['term'], inplace=True)  # Drop the column if it exists

# Calculate VIF for the remaining features
vif_results = calculate_vif(X)
vif_results.head(5)
```

Out[87]:

	Feature	VIF
0	purpose_debt_consolidation	18.37
1	open_acc	13.64
2	total_acc	12.65
3	revol_util	9.04
4	annual_inc	8.03

In [88]:
```
if 'purpose_debt_consolidation' in X.columns:
    X.drop(columns=['purpose_debt_consolidation'], inplace=True)  # Drop the

# Calculate VIF for the remaining features
vif_results = calculate_vif(X)
vif_results.head(5)
```

Out[88]:

	Feature	VIF
0	open_acc	13.09
1	total_acc	12.64
2	revol_util	8.31
3	annual_inc	7.70
4	dti	7.58

In [89]:
```
if 'open_acc' in X.columns:
    X.drop(columns=['open_acc'], inplace=True)  # Drop the column if it exis

# Calculate VIF for the remaining features
vif_results = calculate_vif(X)
vif_results.head(5)
```

Out[89]:

	Feature	VIF
0	total_acc	8.23
1	revol_util	8.00
2	annual_inc	7.60
3	dti	7.02
4	loan_amnt	6.72

K-fold Cross validation

Reduces Variance: Ensures the model is evaluated on multiple data splits, reducing dependence on a single train-test split.

Robust Performance Metrics: Provides a more generalizable estimate of model accuracy.

Utilizes All Data: Every data point is used for both training and testing at least once.

In [91]:
```
X=scaler.fit_transform(X)

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

```
Cross Validation accuracy : 0.891
```

Observation: This indicates that the model achieves an average accuracy of 89.1% across the 5 folds.

SMOTE - Oversampling

In [92]:
```
from imblearn.over_sampling import SMOTE

# Apply SMOTE for oversampling
sm = SMOTE(random_state=42)
X_train_res, y_train_res = sm.fit_resample(X_train, y_train.ravel())

# Print the new shapes of the resampled data
print(f"After OverSampling, the shape of X_train: {X_train_res.shape}")
print(f"After OverSampling, the shape of y_train: {y_train_res.shape}\n")

# Print the counts of each class in the resampled dataset
print(f"After OverSampling, counts of label '1' (minority class): {sum(y_tra
print(f"After OverSampling, counts of label '0' (majority class): {sum(y_tra
```

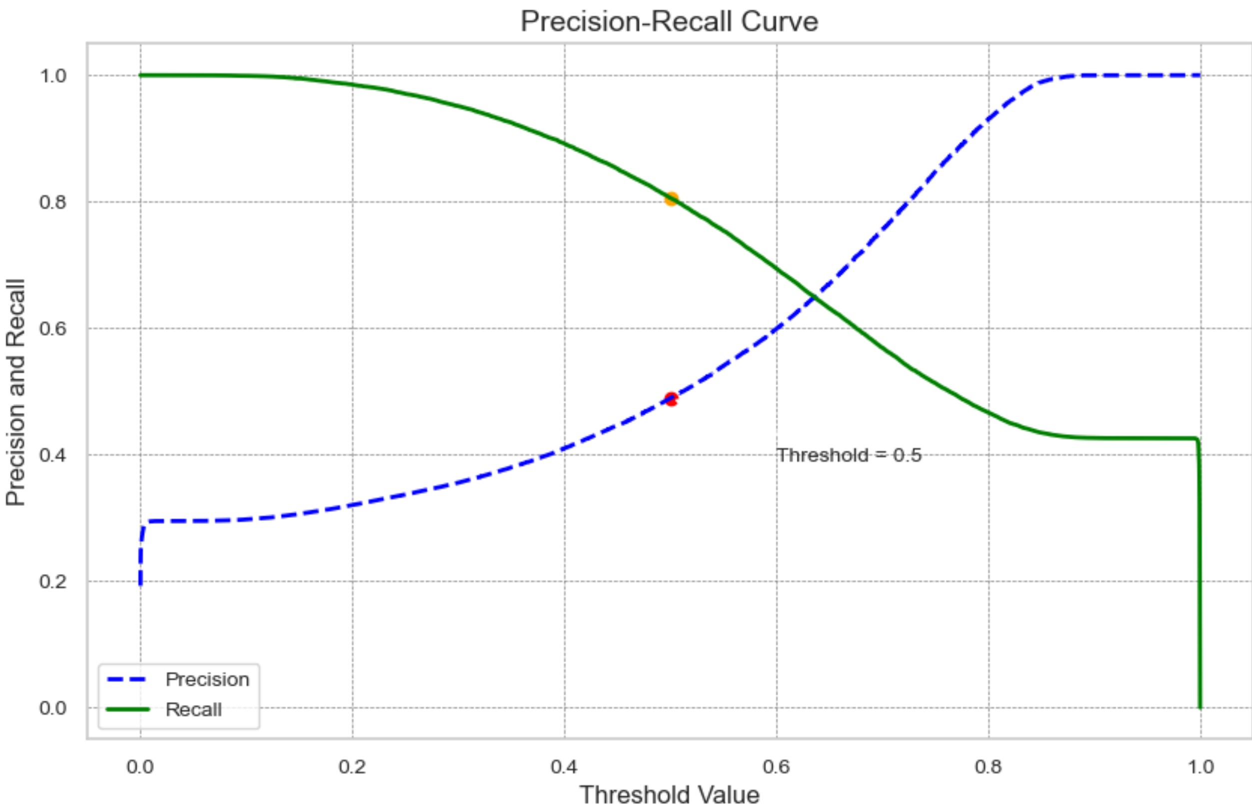
```
After OverSampling, the shape of X_train: (400810, 48)
After OverSampling, the shape of y_train: (400810,)

After OverSampling, counts of label '1' (minority class): 200405
After OverSampling, counts of label '0' (majority class): 200405
```

In [93]:
```
lr1 = LogisticRegression(max_iter=1000)
lr1.fit(X_train_res, y_train_res)
predictions = lr1.predict(X_test)

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

```
              precision    recall  f1-score   support

           0       0.95      0.80      0.87     85888
           1       0.49      0.81      0.61     20468

    accuracy                           0.80    106356
   macro avg       0.72      0.80      0.74    106356
weighted avg       0.86      0.80      0.82    106356
```

Plots the Precision-Recall Curve for the given test labels and predicted probabilities. Parameters: y_test: True binary labels for the test data. pred_proba_c1: Predicted probabilities for the positive class (class 1).

In [94]:
```
def precision_recall_curve_plot(y_test, pred_proba_c1):

    # Compute precision, recall, and thresholds
    precisions, recalls, thresholds = precision_recall_curve(y_test, pred_pr

    # Exclude the last precision/recall value, as it doesn't have a correspo
    thresholds = np.append(thresholds, 1)  # Add a final threshold for plott

    # Create the plot
    plt.figure(figsize=(10, 6))
    plt.plot(thresholds, precisions, linestyle='--', label='Precision', colo
    plt.plot(thresholds, recalls, label='Recall', color='green', linewidth=2

    # Improve axis ticks and labels
    plt.xticks(fontsize=10)
    plt.yticks(fontsize=10)
    plt.xlabel('Threshold Value', fontsize=12)
    plt.ylabel('Precision and Recall', fontsize=12)

    # Add grid, title, and legend
    plt.grid(color='gray', linestyle='--', linewidth=0.5)
    plt.title('Precision-Recall Curve', fontsize=14)
    plt.legend(loc='best', fontsize=10)

    # Optional: Annotate a specific threshold point (e.g., 0.5)
    optimal_threshold = 0.5
    closest_index = np.argmin(np.abs(thresholds - optimal_threshold))
    plt.scatter(thresholds[closest_index], precisions[closest_index], color=
    plt.scatter(thresholds[closest_index], recalls[closest_index], color='or
    plt.annotate('Threshold = 0.5',
                 xy=(thresholds[closest_index], precisions[closest_index]),
                 xytext=(thresholds[closest_index] + 0.1, precisions[closest
                 arrowprops=dict(facecolor='black', arrowstyle='->'), fontsi

    # Show the plot
    plt.show()

# Call the function with your test data and predicted probabilities
precision_recall_curve_plot(y_test, lr1.predict_proba(X_test)[:, 1])
```


Precision-Recall Curve
Precision and Recall
Threshold Value
Threshold = 0.5
Precision
Recall
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

Summary

Key Steps in the Analysis

Data Preprocessing Missing values were treated effectively: Variables like mort_acc were imputed using mean values based on total_acc. Redundant or sparse columns (issue_d, emp_title, title, etc.) were dropped. Categorical variables (grade, home_ownership, verification_status, etc.) were converted to dummy variables. Outliers were handled using the 3-sigma rule to improve model robustness. Min-Max scaling was applied to normalize numerical features.

Feature Engineering Strong correlation was identified between loan_amnt and installment, leading to the removal of the latter. Transformation of categorical variables like term and initial_list_status into numerical formats. Binary encoding for some features, such as pub_rec and pub_rec_bankruptcies.

Multicollinearity Check Variance Inflation Factor (VIF) analysis revealed high collinearity among features like open_acc, total_acc, and revol_util. Features with the highest VIF values were iteratively dropped to ensure model stability. Addressing Class Imbalance

SMOTE (Synthetic Minority Oversampling Technique) was applied, balancing the dataset to ensure equal representation of both classes (Fully Paid and Charged Off). Model Training and Validation Data was split into training (70%) and testing (30%) sets with stratification to maintain class distribution. Logistic regression was applied using balanced data. Cross-validation (5-fold) was performed to ensure generalization, yielding an average accuracy of 89.1%.

Model Performance Metrics

Accuracy on Test Data: 89.0%

Confusion Matrix: True Positives (Fully Paid): 85,364 False Positives: 524 False Negatives: 11,132 True Negatives (Charged Off): 9,336

Classification Report: Precision (0): 88%, Recall: 99%, F1-score: 94% Precision (1): 95%, Recall: 46%, F1-score: 62%

ROC-AUC Score: High area under the ROC curve indicates strong predictive power.

Insights and Observations

Grade and Sub-grade Analysis: Loans graded B (especially sub-grade B3) had the highest likelihood of being fully paid.

Loan Term: Loans with a 36-month term showed higher repayment rates compared to 60-month loans.

Purpose of Loan: Debt consolidation loans were the most common and had relatively better repayment trends.

Employment Length: Individuals with longer employment tenures (10+ years) exhibited higher repayment probabilities.

SMOTE Impact: Balancing the classes led to improved recall for the minority class (Charged Off) at the cost of a slight drop in overall accuracy.

Recommendations:

Focus on Higher Grades: Strengthen approval policies for borrowers with grades B and above, as they show better repayment behavior.

Shorter Loan Terms: Encourage 36-month loans over longer terms to reduce default risk.

Tailored Strategies for Class Imbalance: Incorporate cost-sensitive learning techniques alongside SMOTE to better balance precision and recall for minority classes. This analysis highlights the effective use of logistic regression and data preprocessing techniques to derive actionable insights for better risk management and decision-making.

In [ ]:

```

```