In [1]:
```python
# Importing the necessary libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```python
# converting data into dataframe

loantap = pd.read_csv("logistic_regression.csv")
```

In [3]:
```python
# making an copy of the dataset

df = loantap.copy()
```

Identification of variables

In [4]:
```python
# Top 5 rows of the dataframe

df.head()
```

Out[4]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	hom
0	10000.0	36 months	11.44	329.48	B	B4	Marketing	10+ years	
1	8000.0	36 months	11.99	265.68	B	B5	Credit analyst	4 years	
2	15600.0	36 months	10.49	506.97	B	B3	Statistician	< 1 year	
3	7200.0	36 months	6.49	220.65	A	A2	Client Advocate	6 years	
4	24375.0	60 months	17.27	609.33	C	C5	Destiny Management Inc.	9 years	

5 rows × 27 columns

In [5]:
```python
df.shape
```

Out[5]: (396030, 27)

In [6]:
```
# data info

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
```

In [7]:
```python
# Checking of null values

df.isna().sum()
```

Out[7]:
```
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                    1756
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 [8]:
```python
# Percentage of null values in each columns

df.isna().sum()/len(df)*100
```

Out[8]:
```
loan_amnt               0.000000
term                    0.000000
int_rate                0.000000
installment             0.000000
grade                   0.000000
sub_grade               0.000000
emp_title               5.789208
emp_length              4.621115
home_ownership          0.000000
annual_inc              0.000000
verification_status     0.000000
issue_d                 0.000000
loan_status             0.000000
purpose                 0.000000
title                   0.443401
dti                     0.000000
earliest_cr_line        0.000000
open_acc                0.000000
pub_rec                 0.000000
revol_bal               0.000000
revol_util              0.069692
total_acc               0.000000
initial_list_status     0.000000
application_type        0.000000
mort_acc                9.543469
pub_rec_bankruptcies    0.135091
address                 0.000000
dtype: float64
```

Analysing the basic metrics

In [9]:
```
df.describe(include='all').transpose()
```

Out[9]:

	count	unique	top	freq	mean	st
loan_amnt	396030.0	NaN	NaN	NaN	14113.888089	8357.4413
term	396030	2	36 months	302005	NaN	Na
int_rate	396030.0	NaN	NaN	NaN	13.6394	4.4721
installment	396030.0	NaN	NaN	NaN	431.849698	250.727
grade	396030	7	B	116018	NaN	Na
sub_grade	396030	35	B3	26655	NaN	Na
emp_title	373103	173105	Teacher	4389	NaN	Na
emp_length	377729	11	10+ years	126041	NaN	Na
home_ownership	396030	6	MORTGAGE	198348	NaN	Na
annual_inc	396030.0	NaN	NaN	NaN	74203.175798	61637.6211
verification_status	396030	3	Verified	139563	NaN	Na
issue_d	396030	115	Oct-2014	14846	NaN	Na
loan_status	396030	2	Fully Paid	318357	NaN	Na
purpose	396030	14	debt_consolidation	234507	NaN	Na
title	394274	48816	Debt consolidation	152472	NaN	Na
dti	396030.0	NaN	NaN	NaN	17.379514	18.0190
earliest_cr_line	396030	684	Oct-2000	3017	NaN	Na
open_acc	396030.0	NaN	NaN	NaN	11.311153	5.1376
pub_rec	396030.0	NaN	NaN	NaN	0.178191	0.5306
revol_bal	396030.0	NaN	NaN	NaN	15844.539853	20591.8361
revol_util	395754.0	NaN	NaN	NaN	53.791749	24.4521
total_acc	396030.0	NaN	NaN	NaN	25.414744	11.8869
initial_list_status	396030	2	f	238066	NaN	Na
application_type	396030	3	INDIVIDUAL	395319	NaN	Na
mort_acc	358235.0	NaN	NaN	NaN	1.813991	2.147
pub_rec_bankruptcies	395495.0	NaN	NaN	NaN	0.121648	0.3561
address	396030	393700	USCGC Smith\r\nFPO AE 70466	8	NaN	Na

Insights

Outliers: The significant differences between mean & median in key attributes like loan amount and revolving balance indicate potential outliers.

Loan Duration Preference: A preference for 36-month loan terms among borrowers suggests a balance between manageable installments.

Home Ownership Trends: The prevalence of applicants with mortgaged homes suggests financial stability or a need for substantial, property-secured loans.

Successful Loan Repayment: Most loans being fully paid off reflects positively on borrowers' financial commitment, indicating effective lending criteria.

Debt Consolidation Dominance: The primary use of loans for debt consolidation highlights a common strategy to manage or reduce high-interest debt.

Individual Borrowers: The predominance of individual applicants suggests that personal

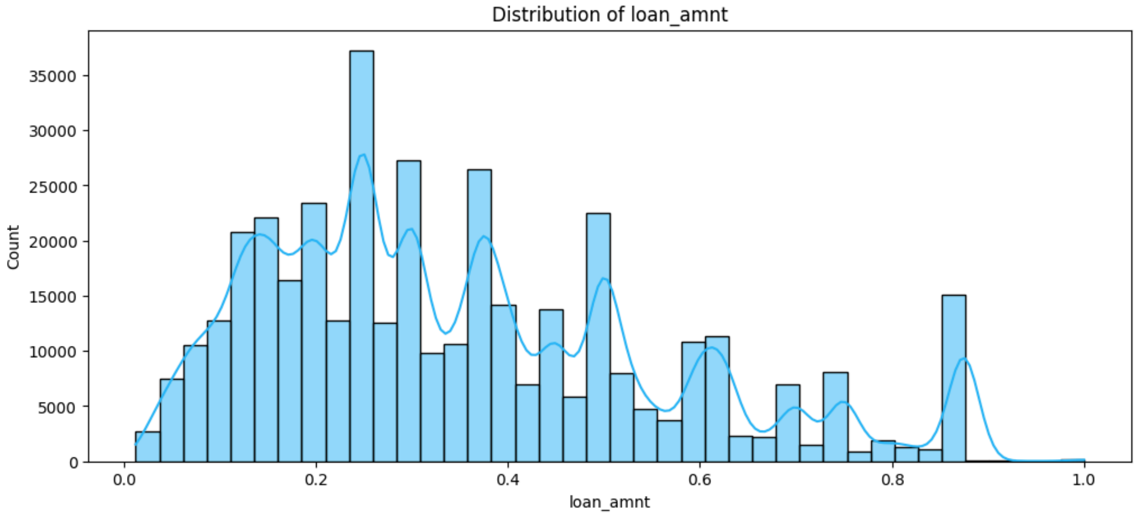
In [10]:

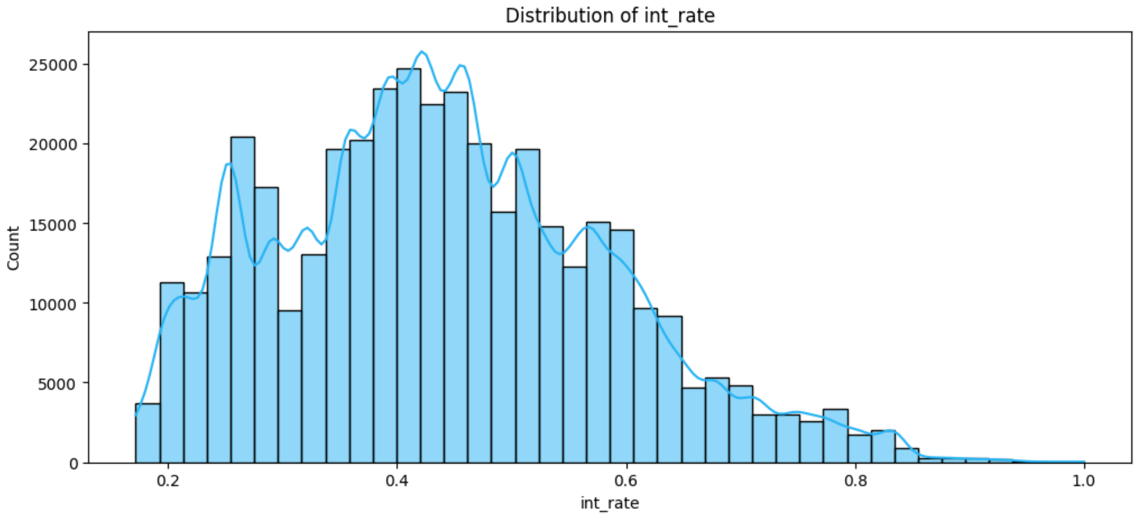
```
n_columns = loantap.select_dtypes('float64').columns.tolist()\

n_columns
```

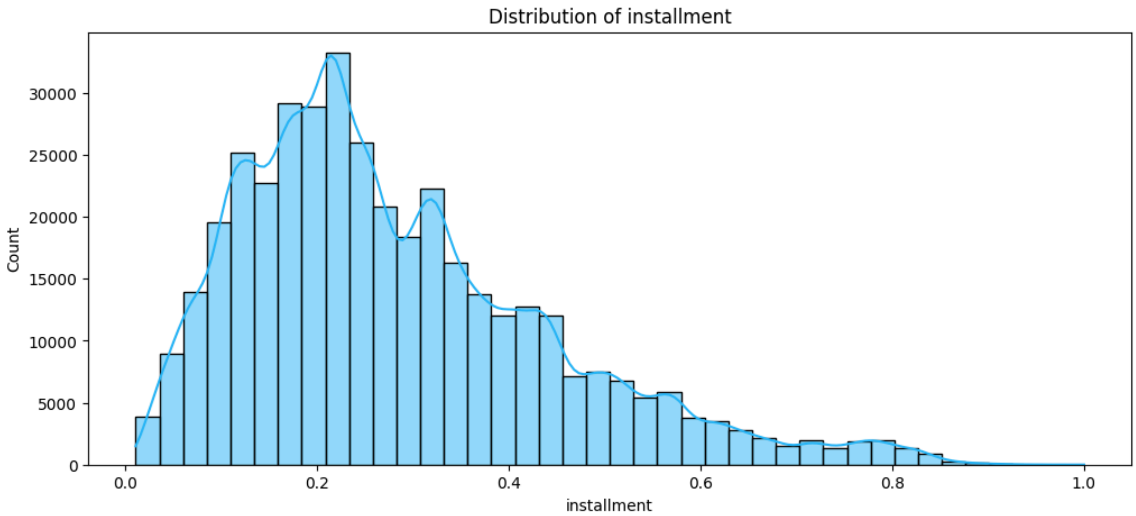
Out[10]:

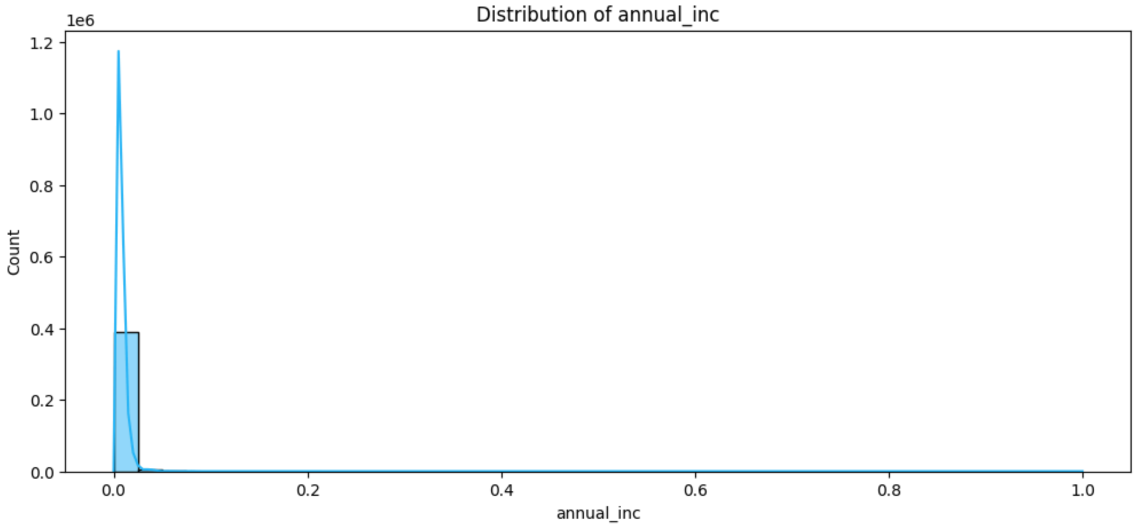
```
['loan_amnt',
 'int_rate',
 'installment',
 'annual_inc',
 'dti',
 'open_acc',
 'pub_rec',
 'revol_bal',
 'revol_util',
 'total_acc',
 'mort_acc',
 'pub_rec_bankruptcies']
```

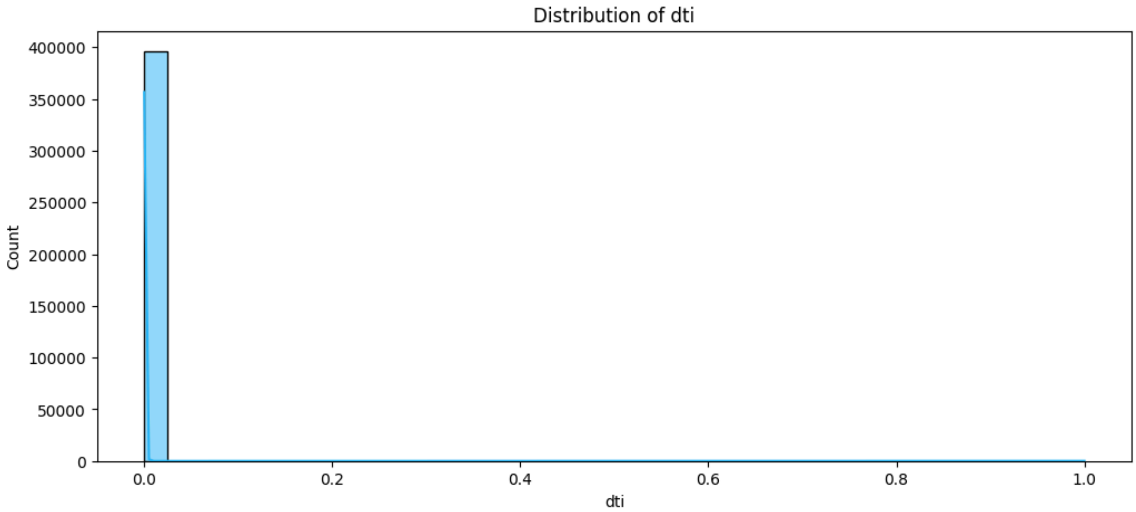
In [11]:
```
for i in n_columns:
    plt.figure(figsize=(12,5))
    plt.title("Distribution of {}".format(i))
    sns.histplot(df[i]/df[i].max(), kde=True,color="#29B6F6", bins=40)
    plt.show()
```

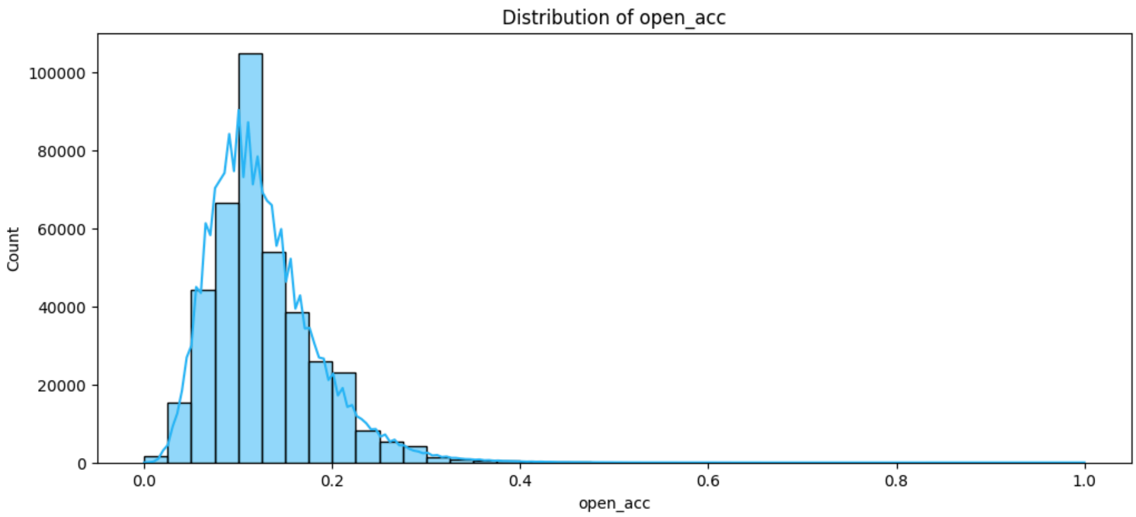


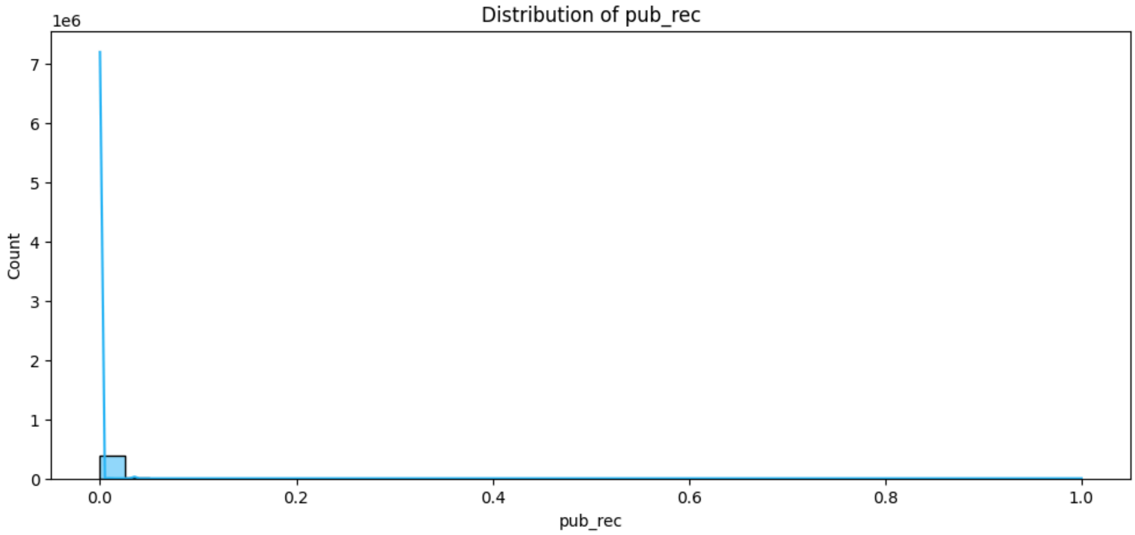


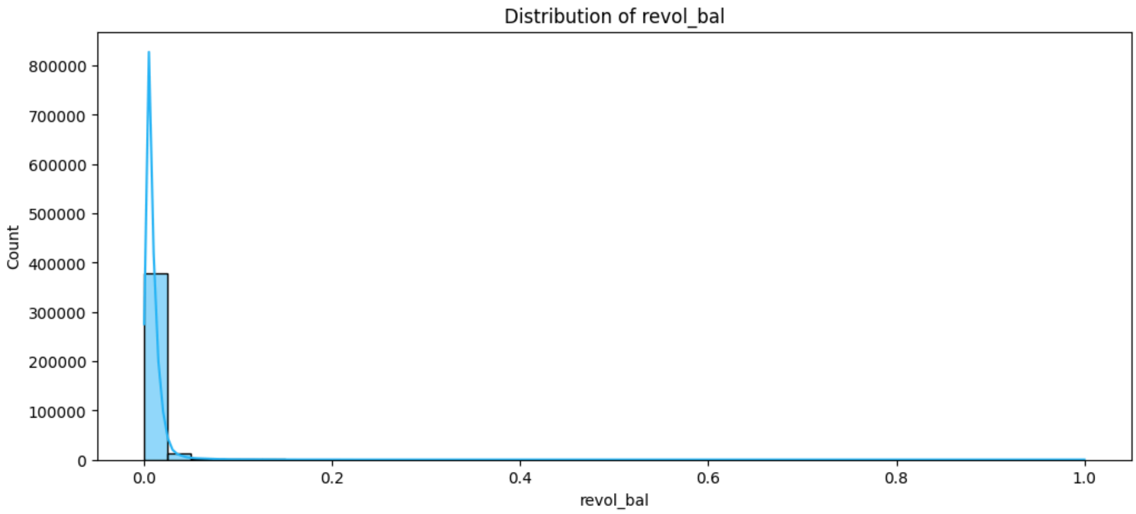


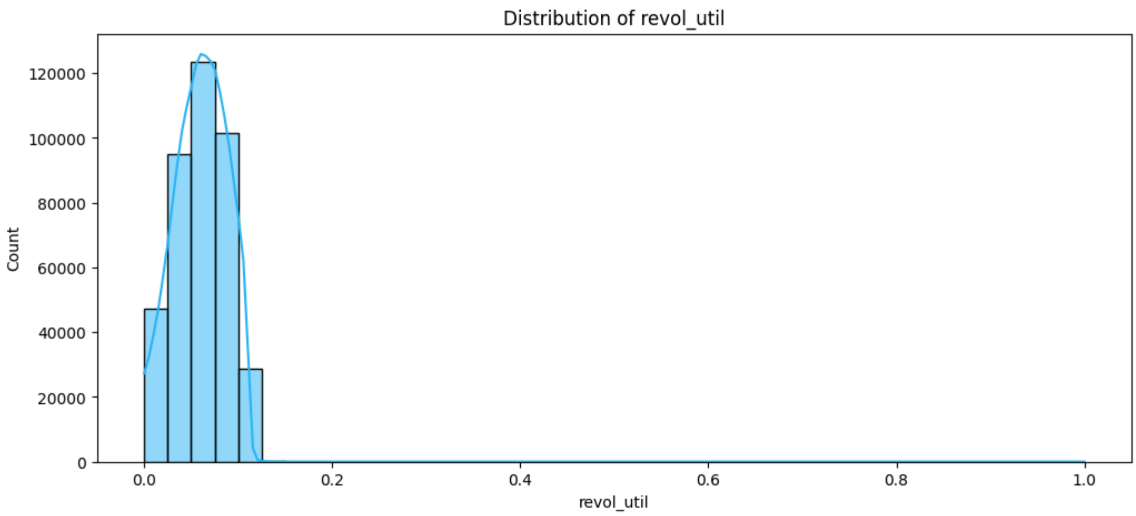

Distribution of annual_inc
1e6
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Count
0.0
0.2
0.4
0.6
0.8
1.0
annual_inc

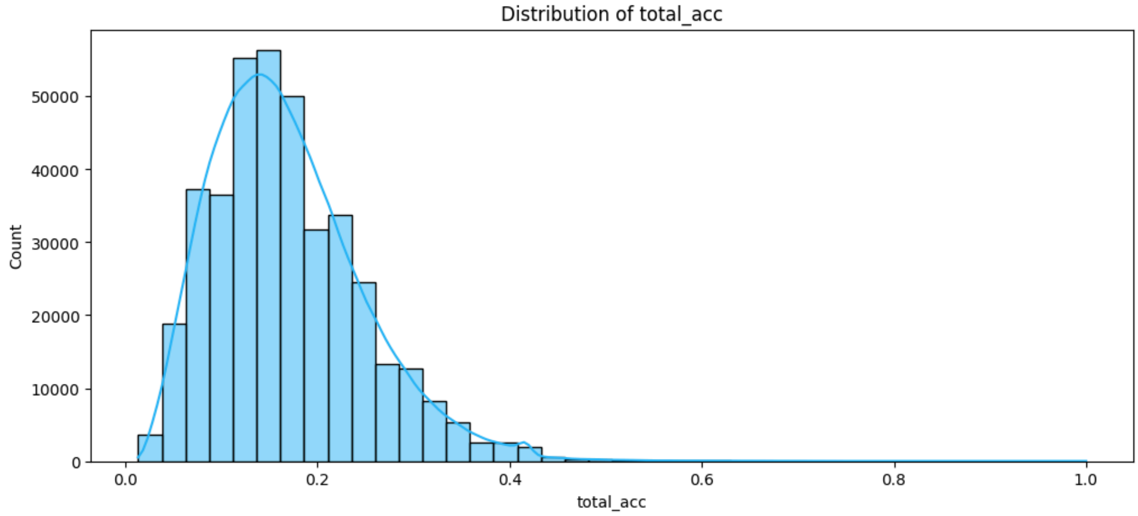

Distribution of dti
400000
350000
300000
250000
200000
150000
100000
50000
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
dti

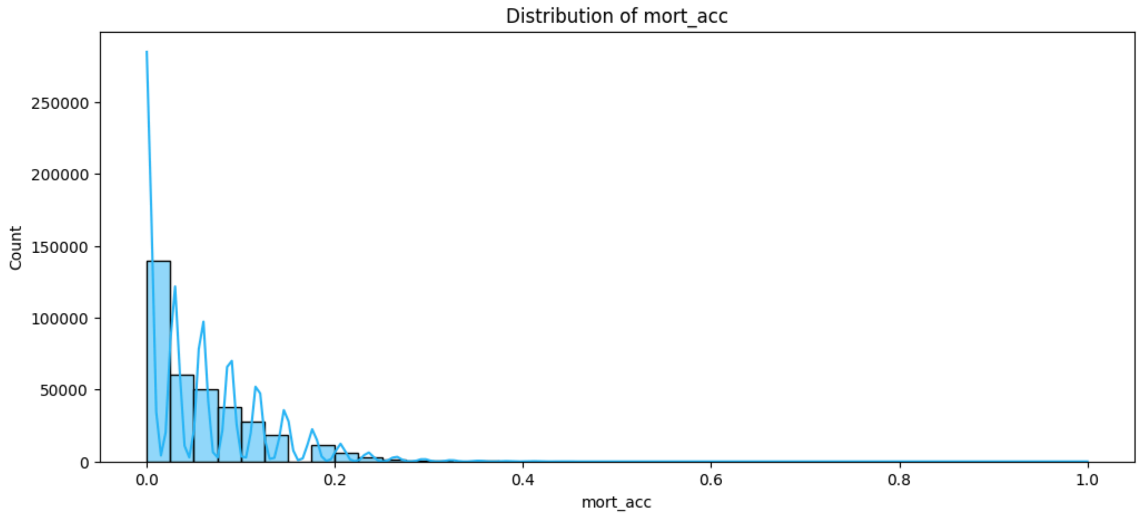

Distribution of open_acc
100000
80000
60000
40000
20000
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
open_acc

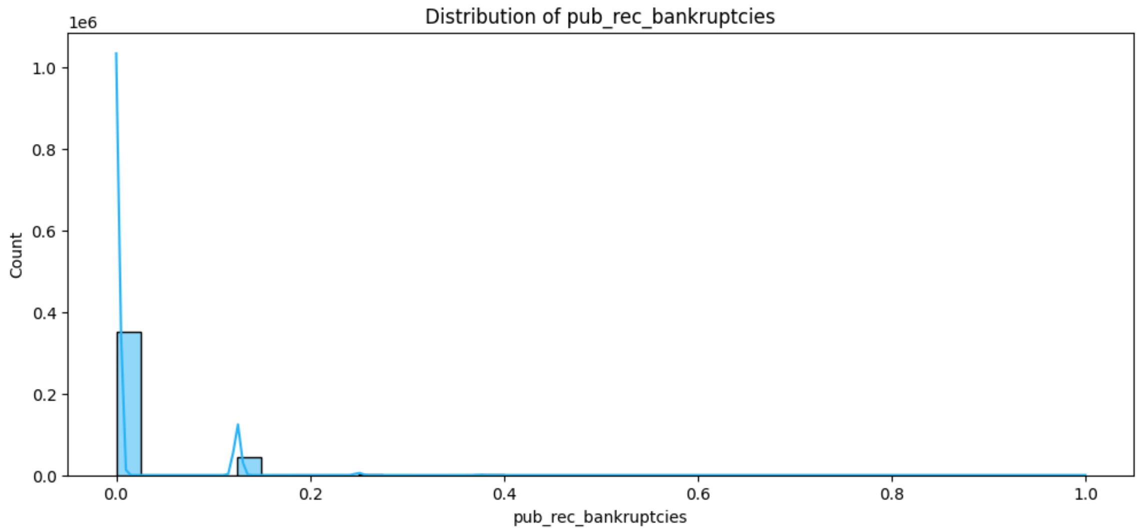

Distribution of pub_rec
1e6
7
6
5
4
3
2
1
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
pub_rec


Distribution of revol_bal
800000
700000
600000
500000
400000
300000
200000
100000
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
revol_bal


Distribution of revol_util
120000
100000
80000
60000
40000
20000
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
revol_util

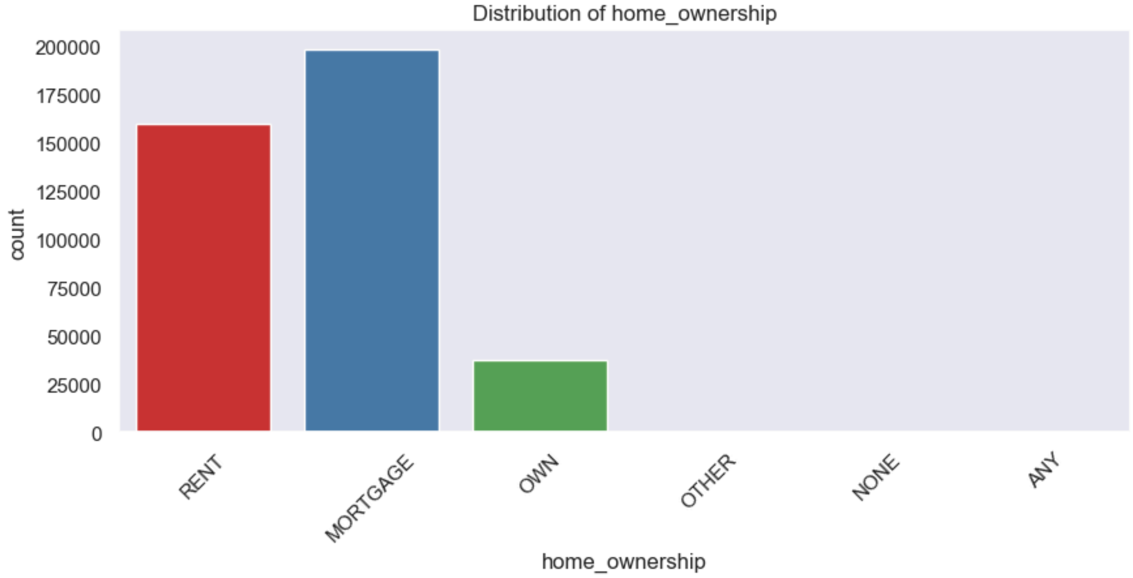


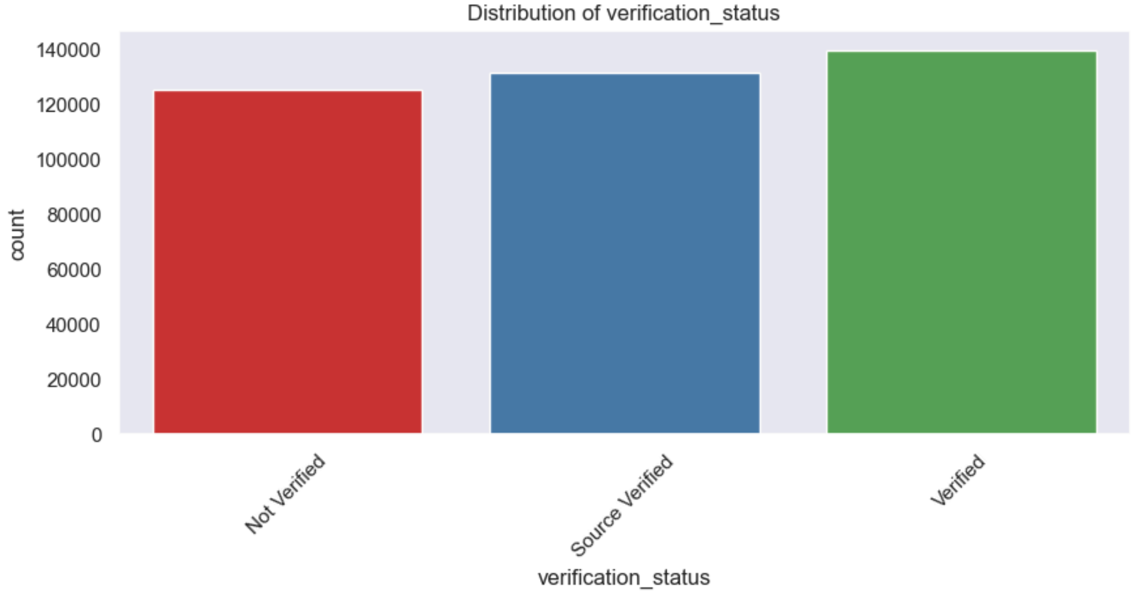


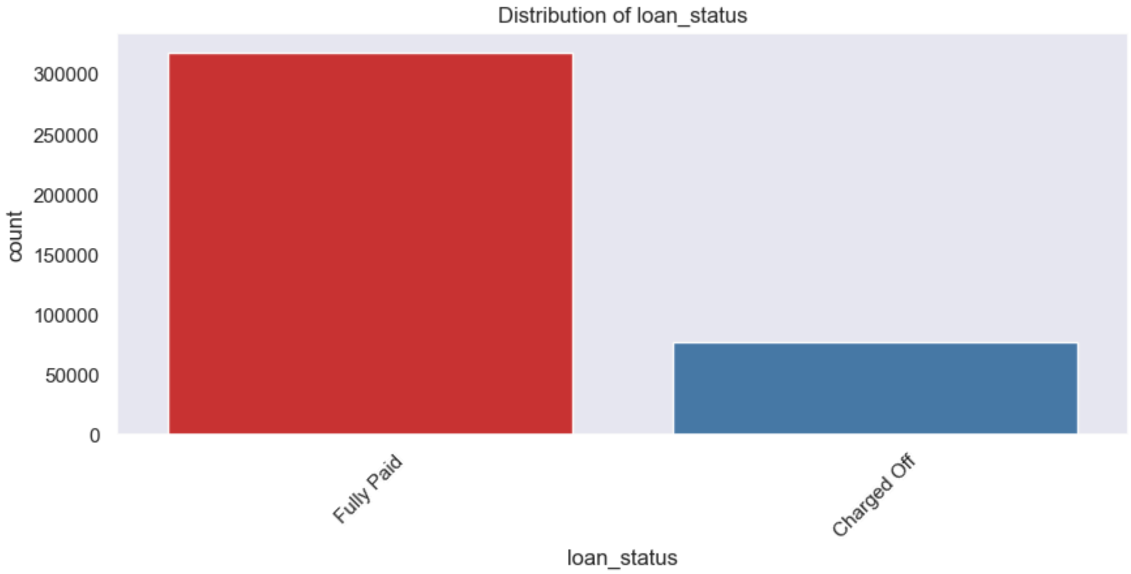


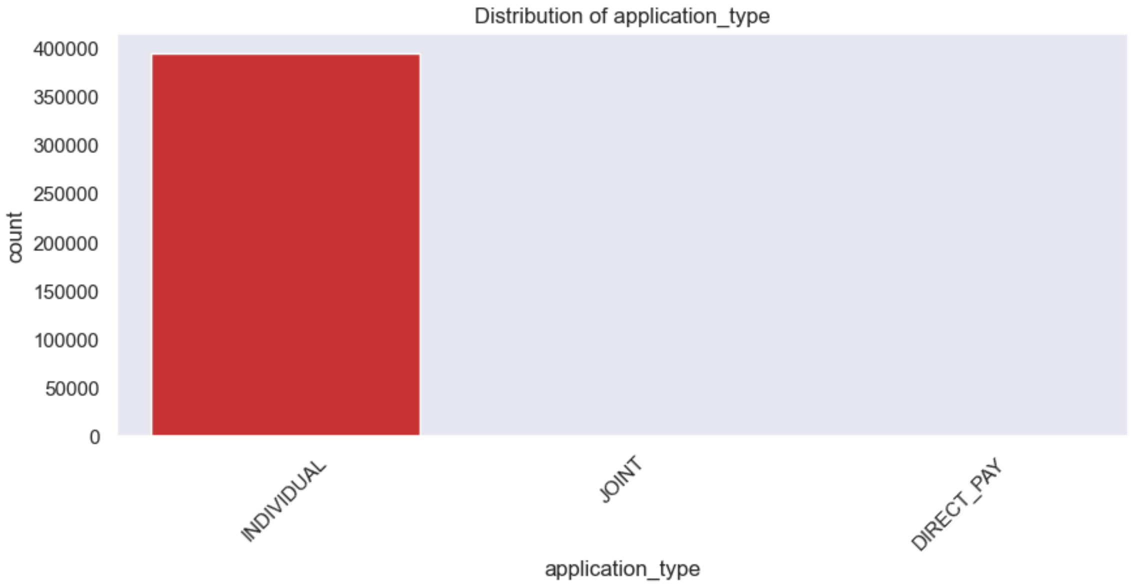
In [12]:
```
c_columns = ['home_ownership', 'verification_status', 'loan_status', 'appli
```

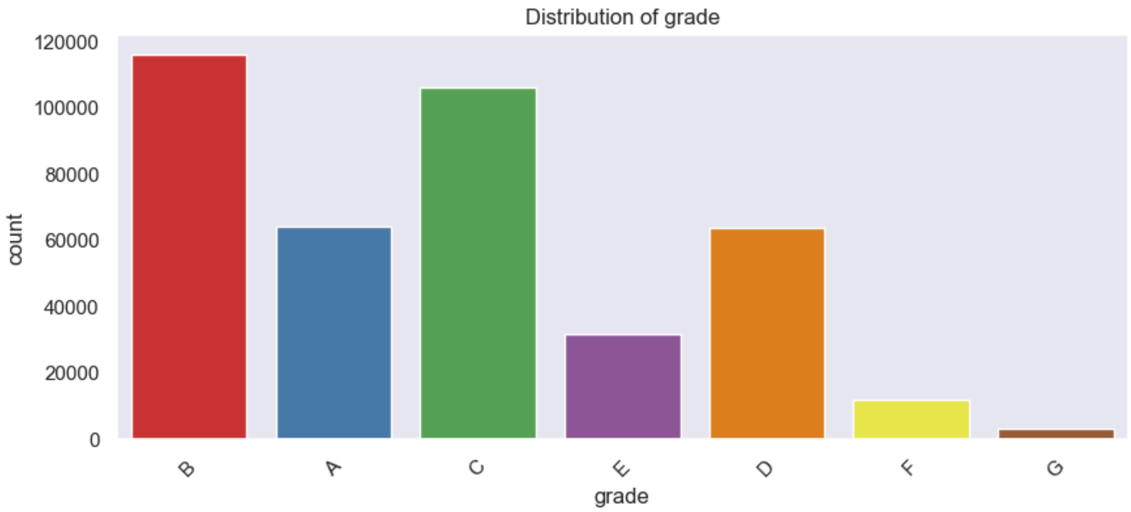
In [13]:
```
custom_palette = sns.color_palette("Set1", 8)

for i in c_columns:
    plt.figure(figsize=(10, 4))
    sns.set(style="dark")
    plt.title(f'Distribution of {i}')
    sns.countplot(data=df, x=i, palette=custom_palette)
    plt.xticks(rotation=45)
    plt.show()
```

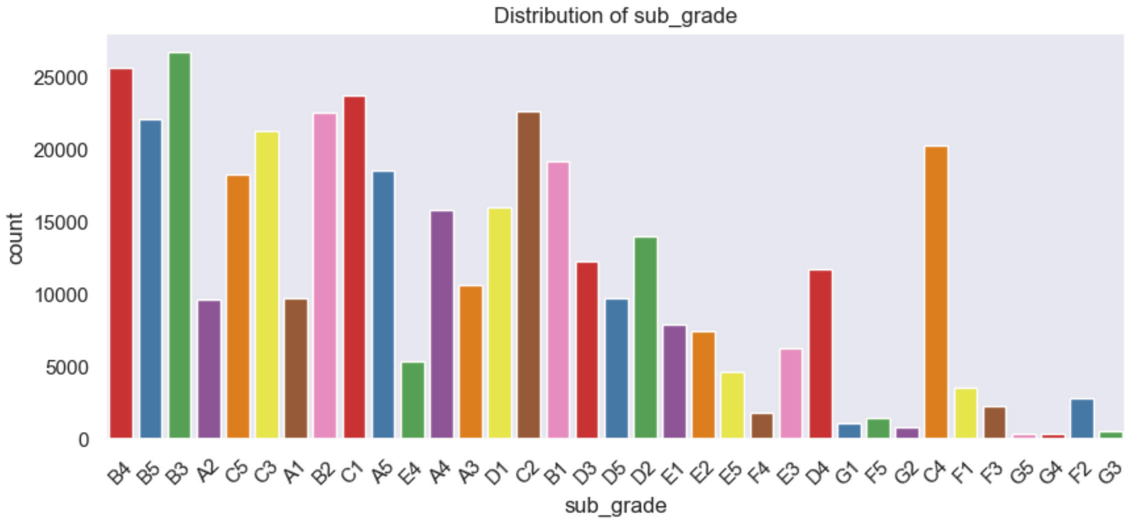


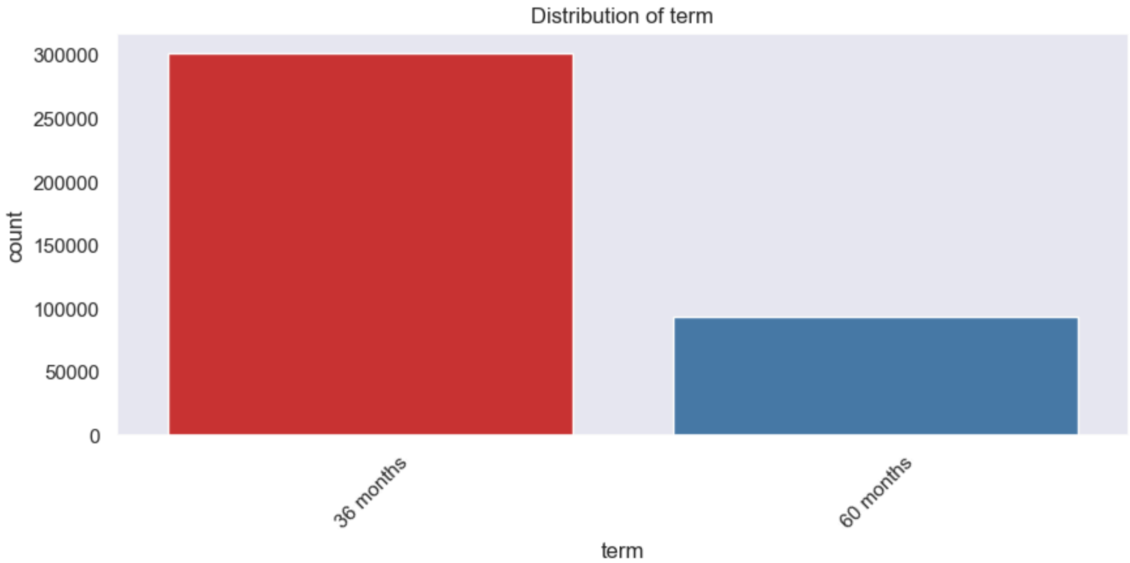



Distribution of loan_status
count
300000
250000
200000
150000
100000
50000
0
Fully Paid
Charged Off
loan_status


Distribution of application_type
count
400000
350000
300000
250000
200000
150000
100000
50000
0
INDIVIDUAL
JOINT
DIRECT_PAY
application_type


Distribution of grade
count
120000
100000
80000
60000
40000
20000
0
B
A
C
E
D
F
G
grade

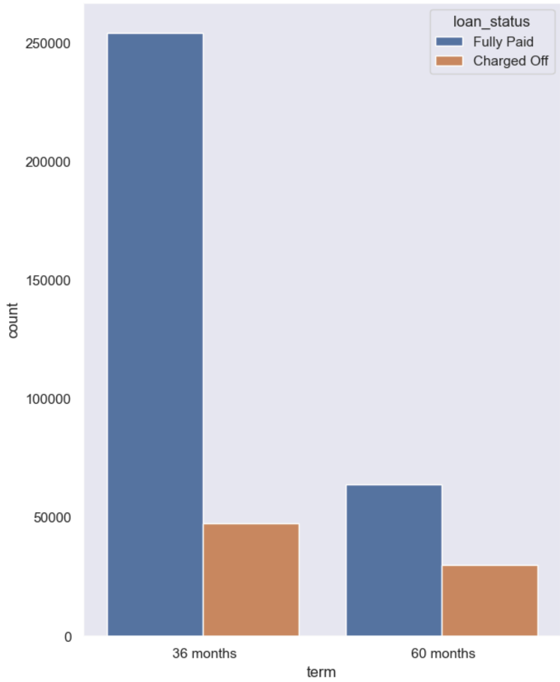

Distribution of sub_grade
count
25000
20000
15000
10000
5000
0
B4 B5 B3 A2 C5 C3 A1 B2 C1 A5 E4 A4 A3 D1 C2 B1 D3 D5 D2 E1 E2 E5 F4 E3 D4 G1 F5 G2 C4 F1 F3 G5 G4 F2 G3
sub_grade

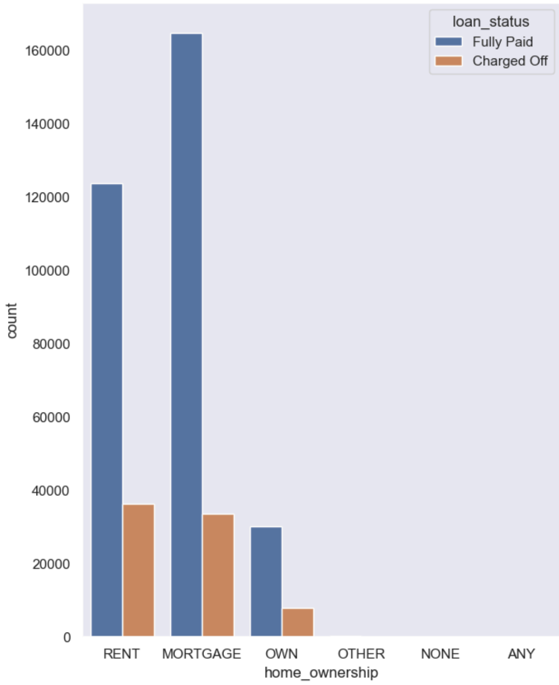

Distribution of term
count
300000
250000
200000
150000
100000
50000
0
36 months
60 months
term

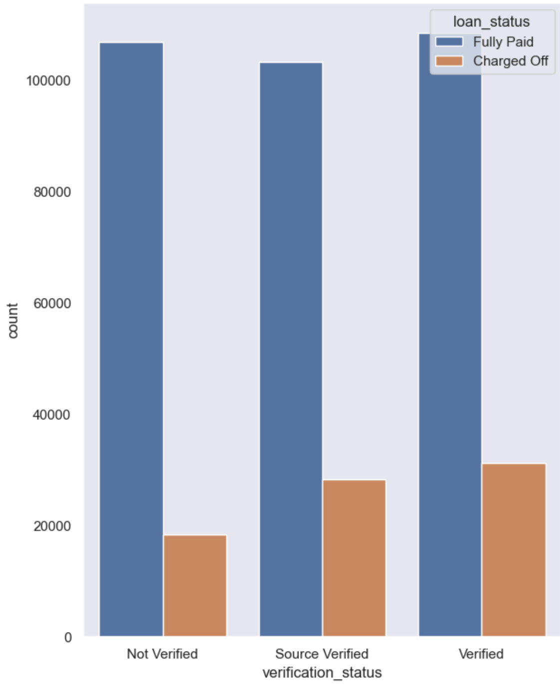
Bivariate Analysis

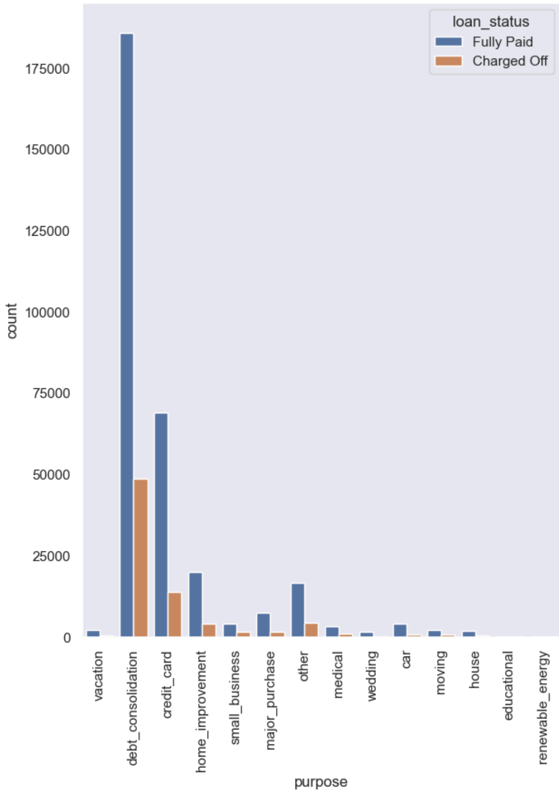
In [14]:

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

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

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

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

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

plt.show()
```

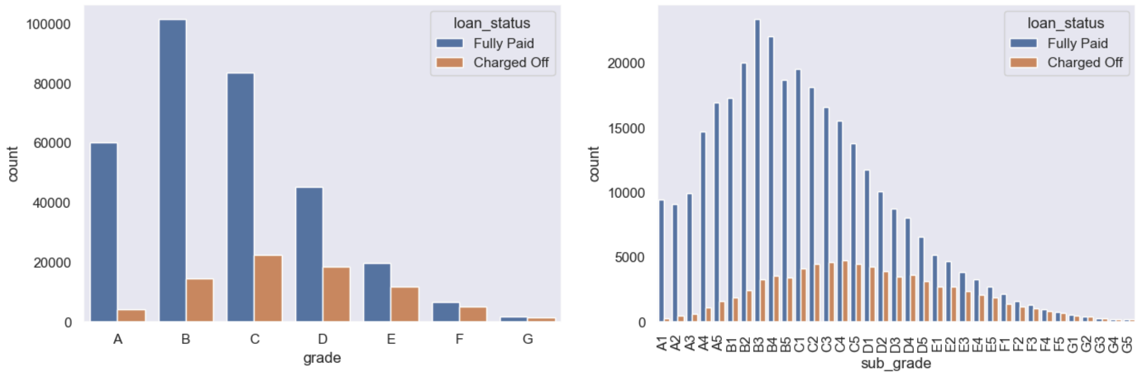

loan_status
Fully Paid
Charged Off
count
0
50000
100000
150000
200000
250000
36 months
60 months
term


loan_status
Fully Paid
Charged Off
count
0
20000
40000
60000
80000
100000
120000
140000
160000
RENT
MORTGAGE
OWN
OTHER
NONE
ANY
home_ownership

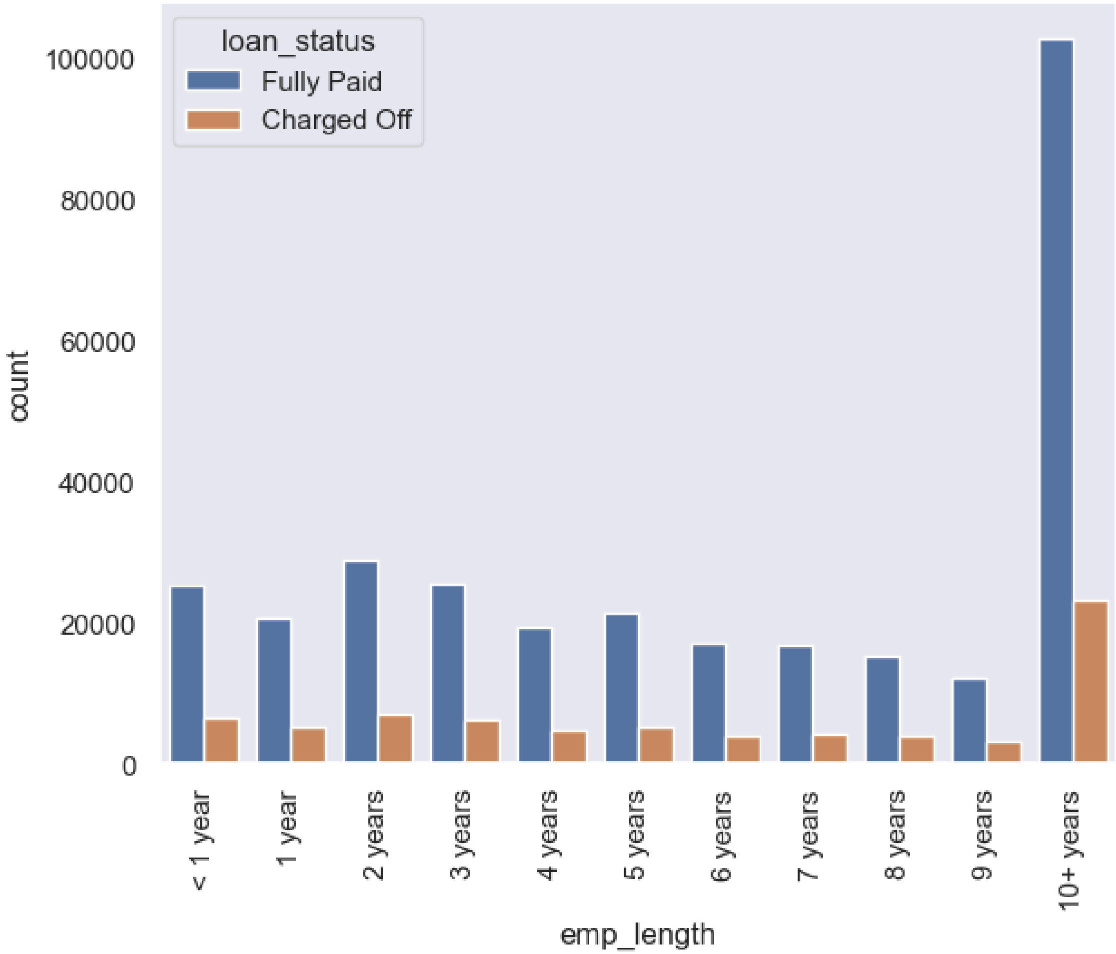

loan_status
Fully Paid
Charged Off
count
0
20000
40000
60000
80000
100000
Not Verified
Source Verified
Verified
verification_status


loan_status
Fully Paid
Charged Off
count
0
25000
50000
75000
100000
125000
150000
175000
vacation
debt_consolidation
credit_card
home_improvement
small_business
major_purchase
other
medical
wedding
car
moving
house
educational
renewable_energy
purpose

In [15]:
```
plt.figure(figsize=(15, 10))

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

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

plt.show()
```



In [16]:
```
plt.figure(figsize=(15,12))

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

plt.show()
```



Insights

Loan Terms: The most popular loan term is 36 months, with a high completion rate.

Loan Types: Mortgages and rental loans are the most common loan types. Debt consolidation loans are also frequently used.

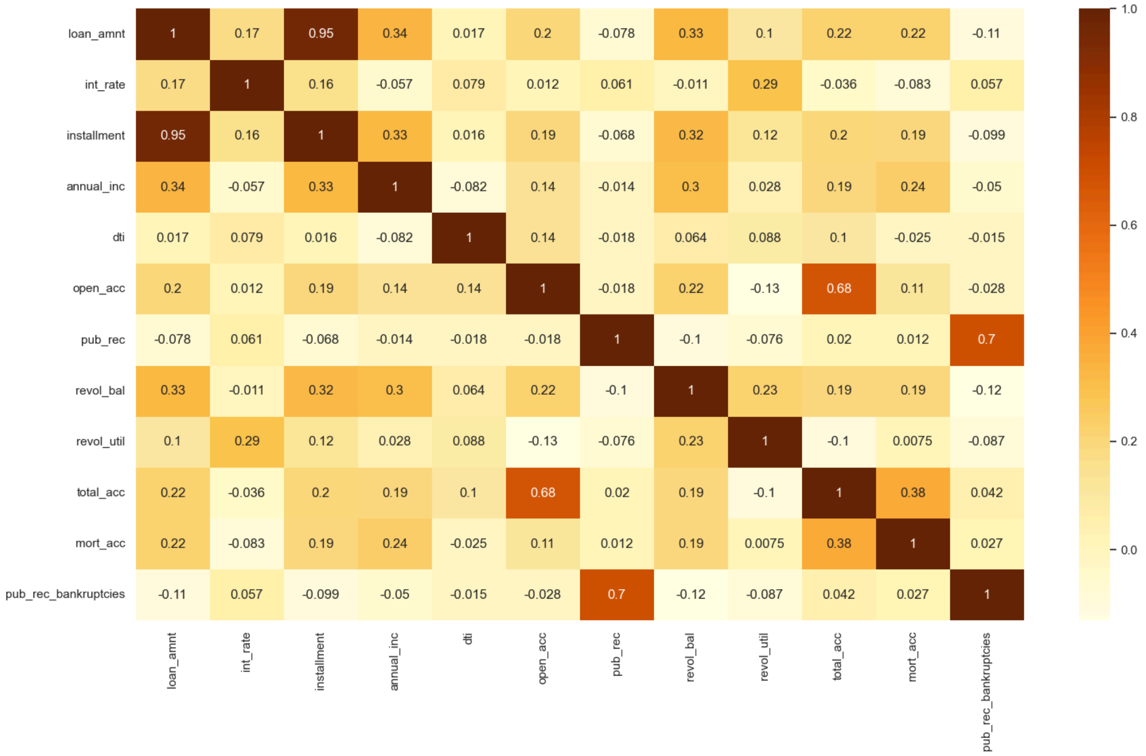
Creditworthiness: Borrowers with a credit grade of "B" and a subgrade of "B3" tend to have the highest repayment rates.

Occupations: Managers and teachers are the professions with the highest loan approval rates.

Repayment: Individuals employed for over 10 years demonstrate a strong track record of loan repayment.

Correlation Analysis

In [17]:
```
plt.figure(figsize=(18,10))
sns.heatmap(df.corr(numeric_only=True), cmap = 'YlOrBr', annot = True)
plt.show()
```



In [18]:
```
df.corr(numeric_only=True)
```

Out[18]:

	loan_amnt	int_rate	installment	annual_inc	dti	open_acc	p
loan_amnt	1.000000	0.168921	0.953929	0.336887	0.016636	0.198556	-0.
int_rate	0.168921	1.000000	0.162758	-0.056771	0.079038	0.011649	0.
installment	0.953929	0.162758	1.000000	0.330381	0.015786	0.188973	-0.
annual_inc	0.336887	-0.056771	0.330381	1.000000	-0.081685	0.136150	-0.
dti	0.016636	0.079038	0.015786	-0.081685	1.000000	0.136181	-0.
open_acc	0.198556	0.011649	0.188973	0.136150	0.136181	1.000000	-0.
pub_rec	-0.077779	0.060986	-0.067892	-0.013720	-0.017639	-0.018392	1.
revol_bal	0.328320	-0.011280	0.316455	0.299773	0.063571	0.221192	-0.
revol_util	0.099911	0.293659	0.123915	0.027871	0.088375	-0.131420	-0.
total_acc	0.223886	-0.036404	0.202430	0.193023	0.102128	0.680728	0.
mort_acc	0.222315	-0.082583	0.193694	0.236320	-0.025439	0.109205	0
pub_rec_bankruptcies	-0.106539	0.057450	-0.098628	-0.050162	-0.014558	-0.027732	0.

Insights:

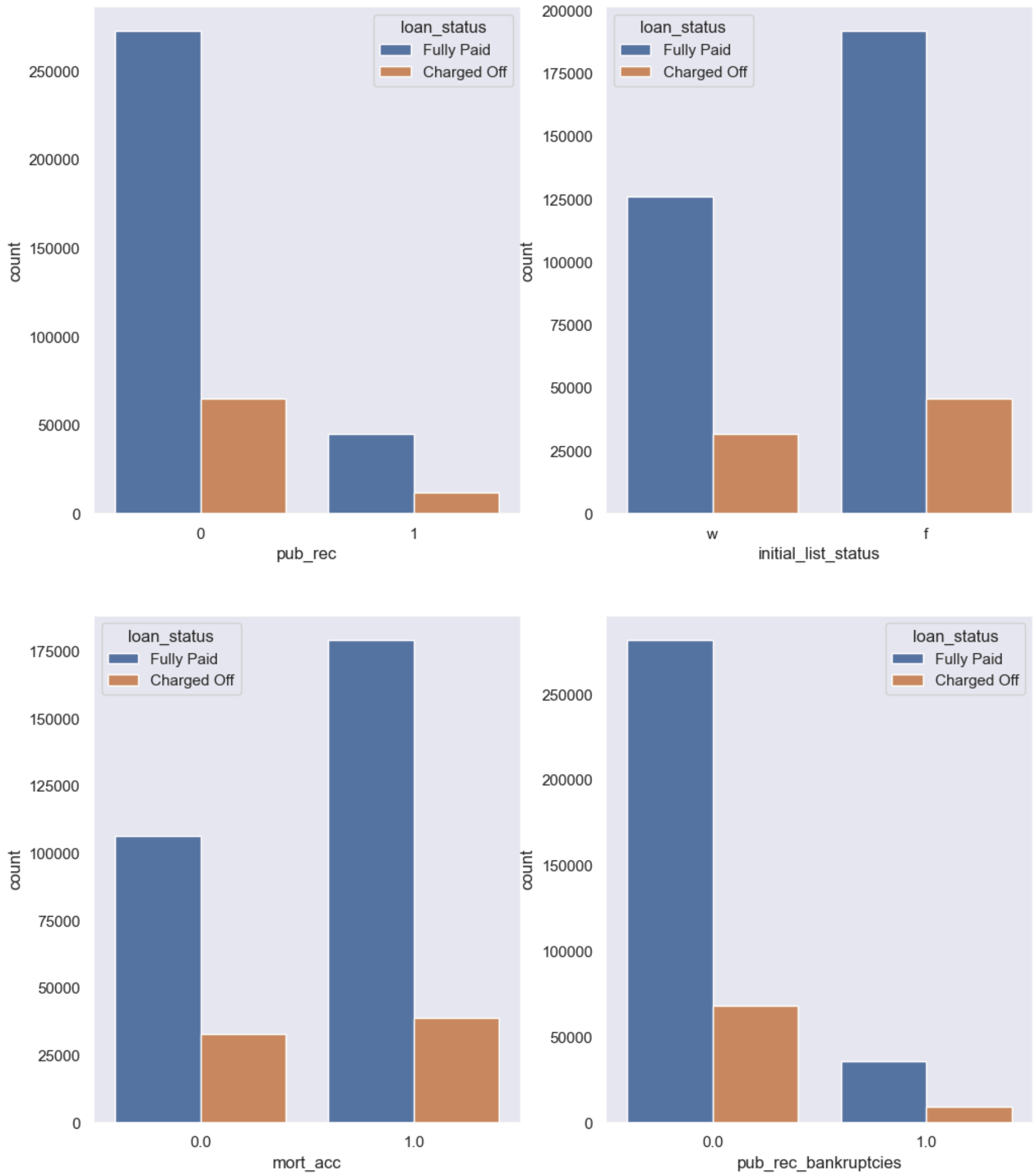
- Positive correlation with annual income (annual_inc) - Higher income allows for larger loan applications.
- Weak positive correlation with installment amount (installment) - Makes sense as larger loans will typically have higher installments.
- Weak positive correlation with total accounts (total_acc) and mortgage accounts (mort_acc) - Borrowers with more established credit lines may be eligible for higher loan amounts.
- Weak negative correlation with annual income (annual_inc) - Generally, borrowers with higher income qualify for lower interest rates.
- Weak positive correlation with total accounts (total_acc) and mortgage accounts (mort_acc) - People with a higher income may tend to have more credit accounts.
- Positive correlation between revolving balance (revol_bal) and credit line utilization (revol_util) - This indicates that people with higher credit balances also tend to have a higher utilization ratio.
- Weak positive correlation between number of open accounts (open_acc) and total accounts (total_acc) - As expected, people with more open accounts tend to have more total accounts.

Data Preprocessing using Feautre Engineering

In [19]:
```
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 [20]:
```
df['pub_rec']=df.pub_rec.apply(pub_rec)

df['mort_acc']=df.mort_acc.apply(mort_acc)

df['pub_rec_bankruptcies']=df.pub_rec_bankruptcies.apply(pub_rec_bankruptci
```

In [21]:
```
plt.figure(figsize=(12,30))

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

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

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

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

plt.show()
```



Most the loan disbursed to the people who do not hold bankrupties record have successfully paid loan

Duplicate checks

In [22]:
```
df.duplicated().sum()
```

Out[22]:
```
0
```

Missing values

In [23]:
```
df.isnull().sum()
```

Out[23]:
```
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                    1756
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 [24]:
```
numeric_columns = df.select_dtypes(include=['float64', 'int64'])

total_acc_avg = numeric_columns.groupby('total_acc')['mort_acc'].mean()

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_a
```

In [25]:
```
df.isnull().sum()
```

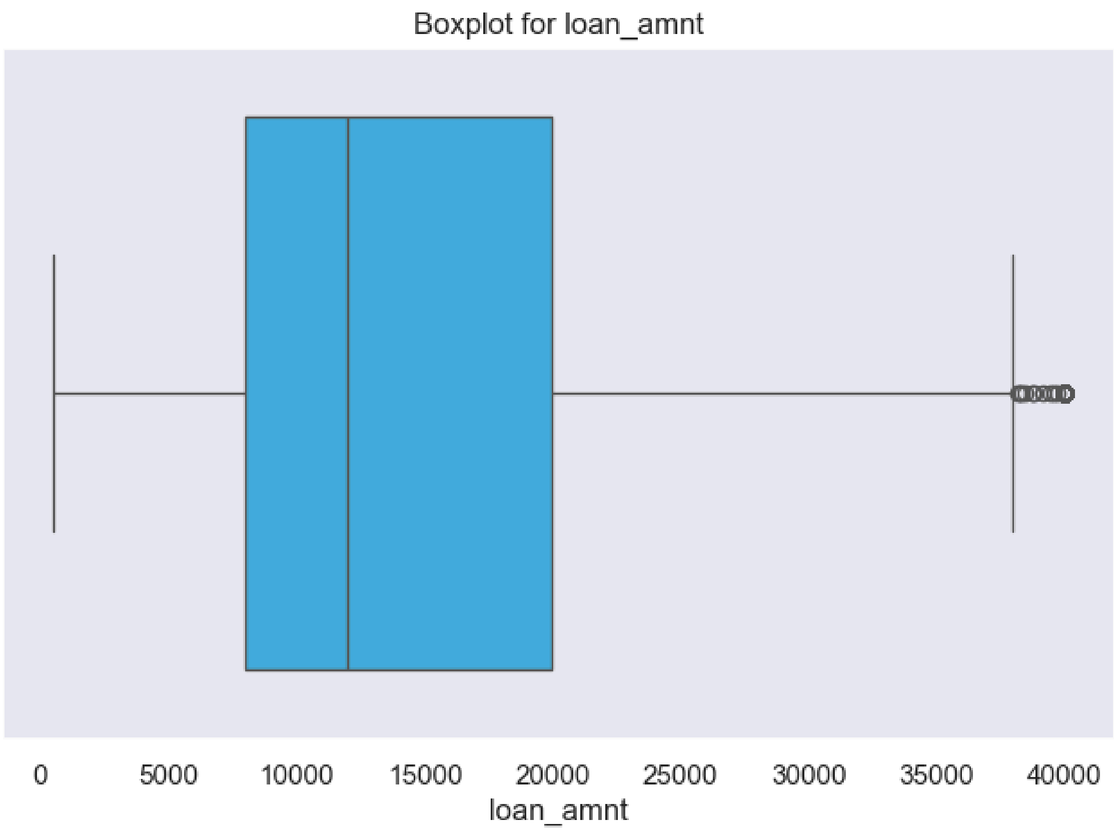
Out[25]:
```
loan_amnt                   0
term                        0
int_rate                    0
installment                 0
grade                       0
sub_grade                   0
emp_title               22927
emp_length              18301
home_ownership              0
annual_inc                  0
verification_status         0
issue_d                     0
loan_status                 0
purpose                     0
title                    1756
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                    0
pub_rec_bankruptcies      535
address                     0
dtype: int64
```

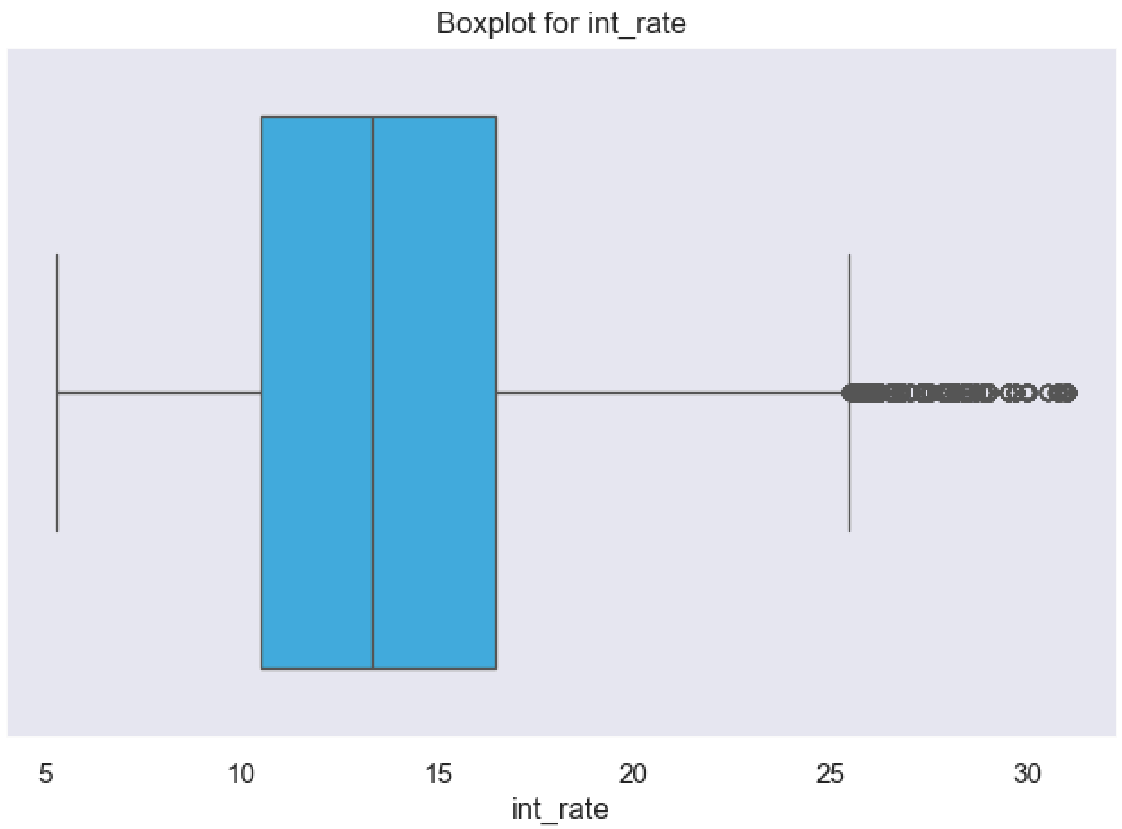
In [26]:
```
# droping remaining null values

df.dropna(inplace=True)

df.shape
```

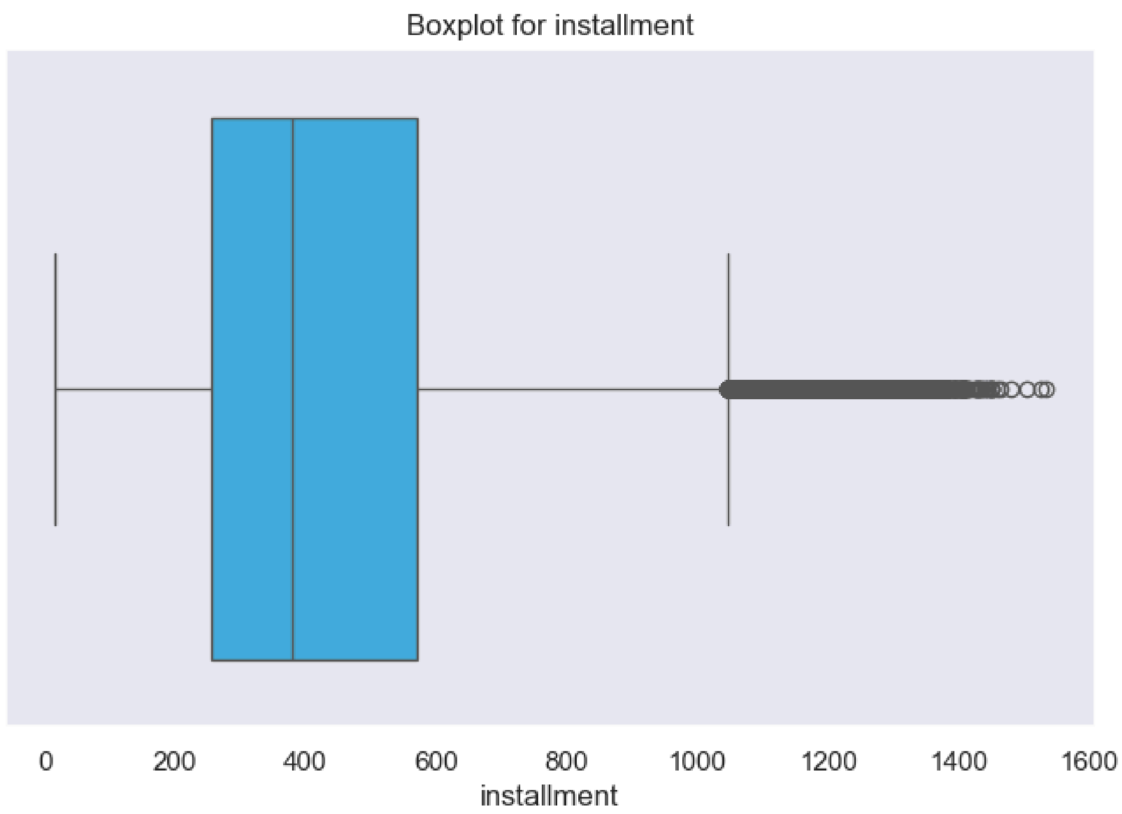
Out[26]:
```
(370621, 27)
```

Outlier Detection

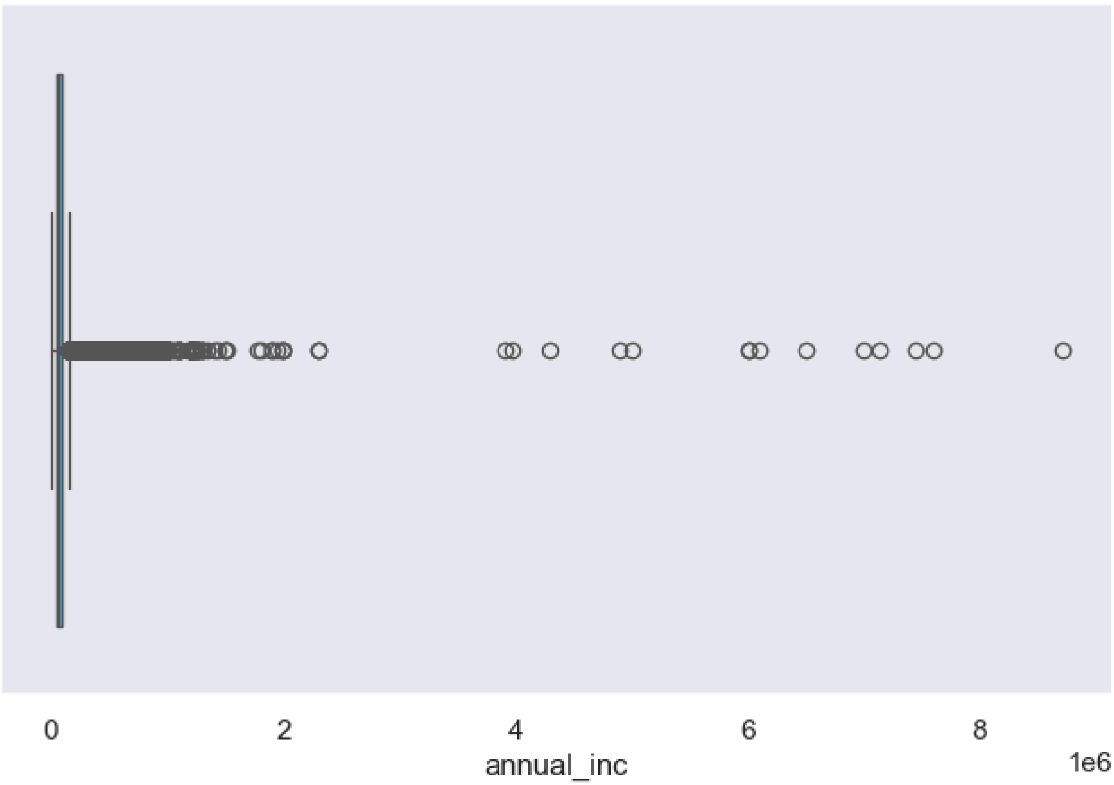
In [27]:
```
def box_plot(col):
    if col in df.columns:
        plt.figure(figsize=(8, 5))
        sns.boxplot(x=df[col],color="#29B6F6")
        plt.title('Boxplot for {}'.format(col))
        plt.show()
    else:
        print(f"Column '{col}' not found in the DataFrame.")

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



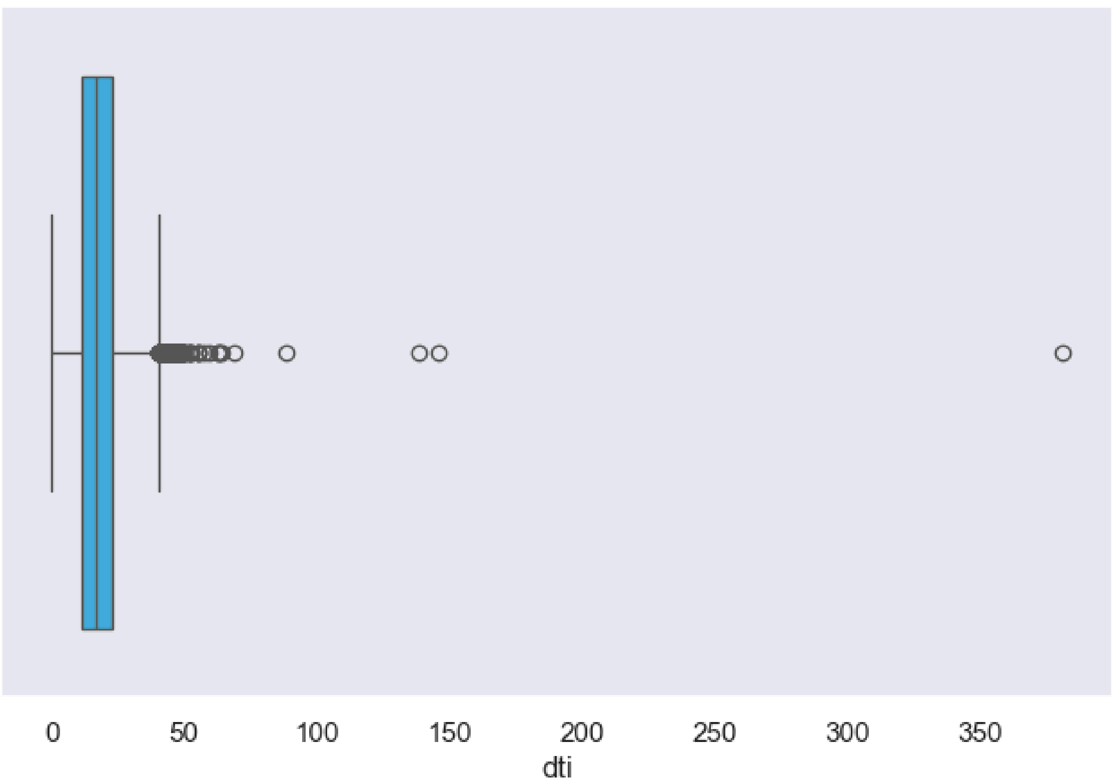

Boxplot for int_rate
5
10
15
20
25
30
int_rate


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

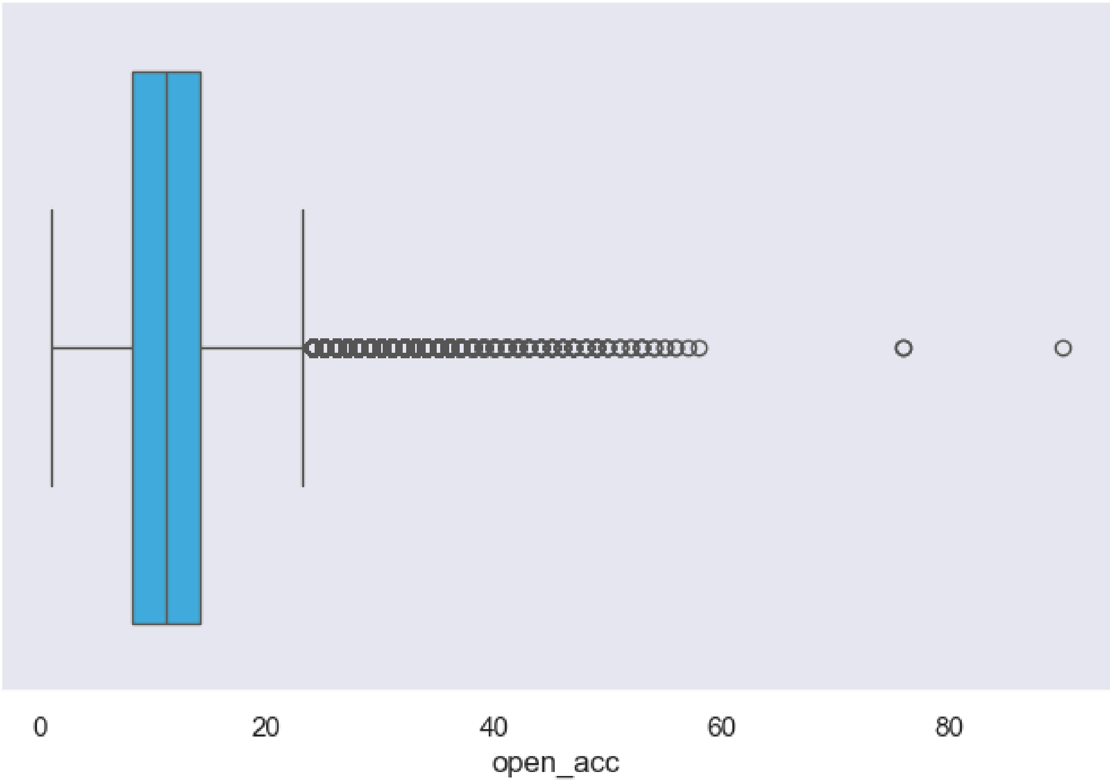
Boxplot for annual_inc



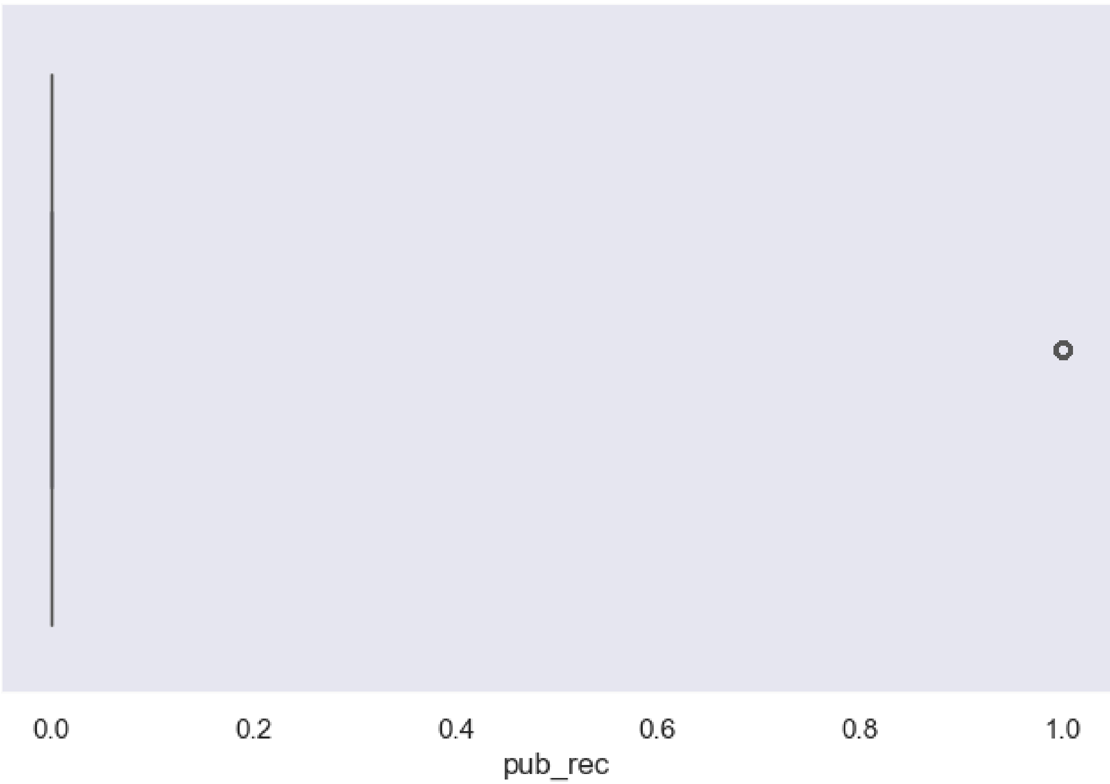
Boxplot for dti

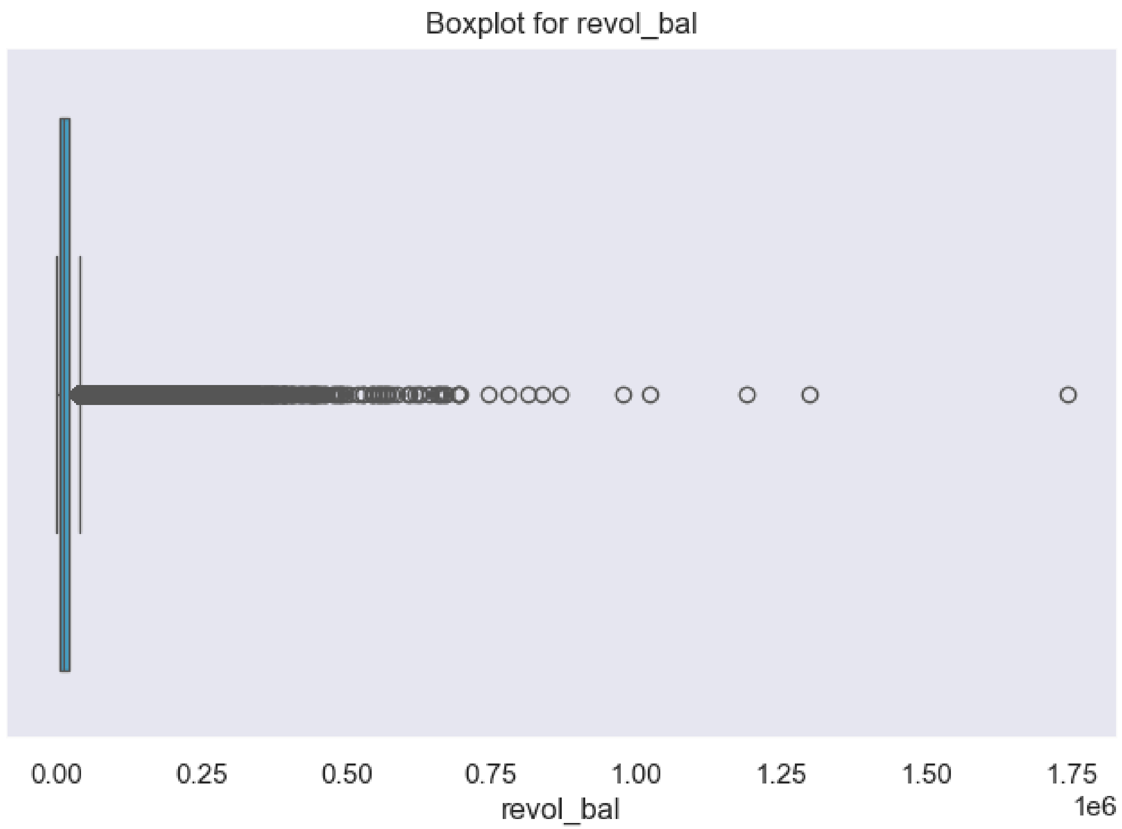


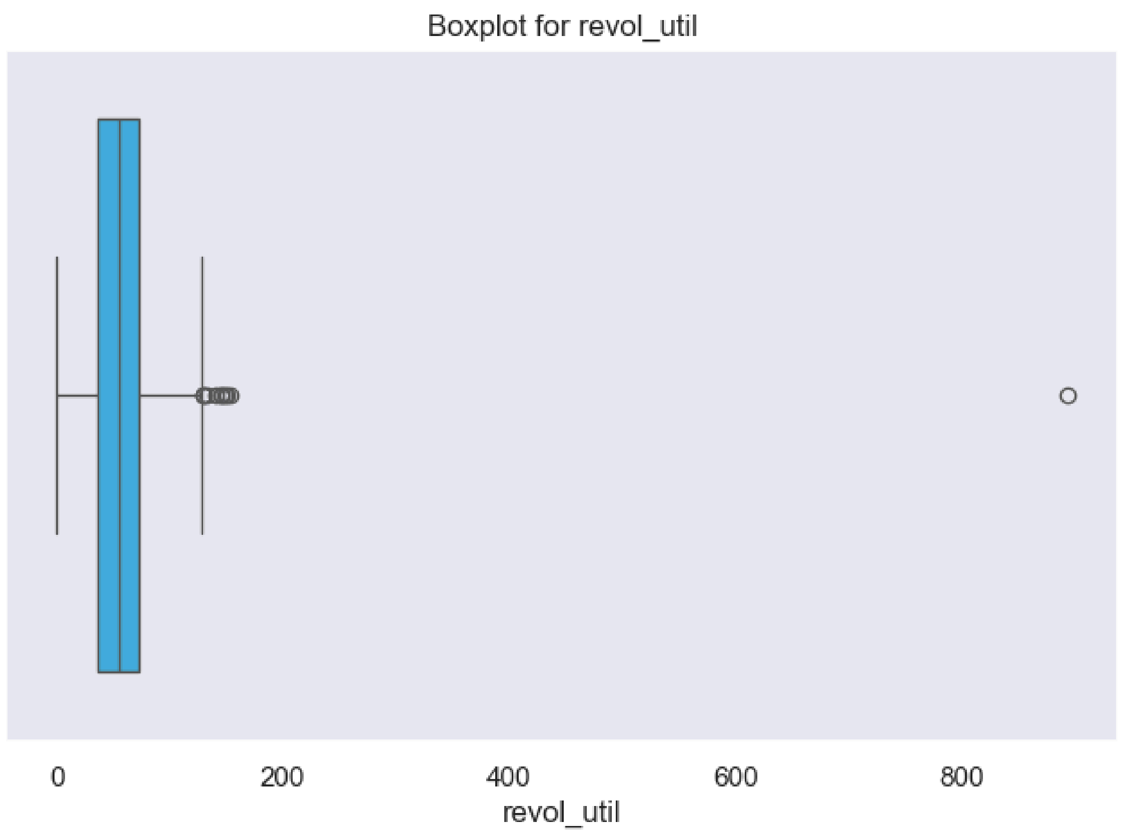
Boxplot for open_acc



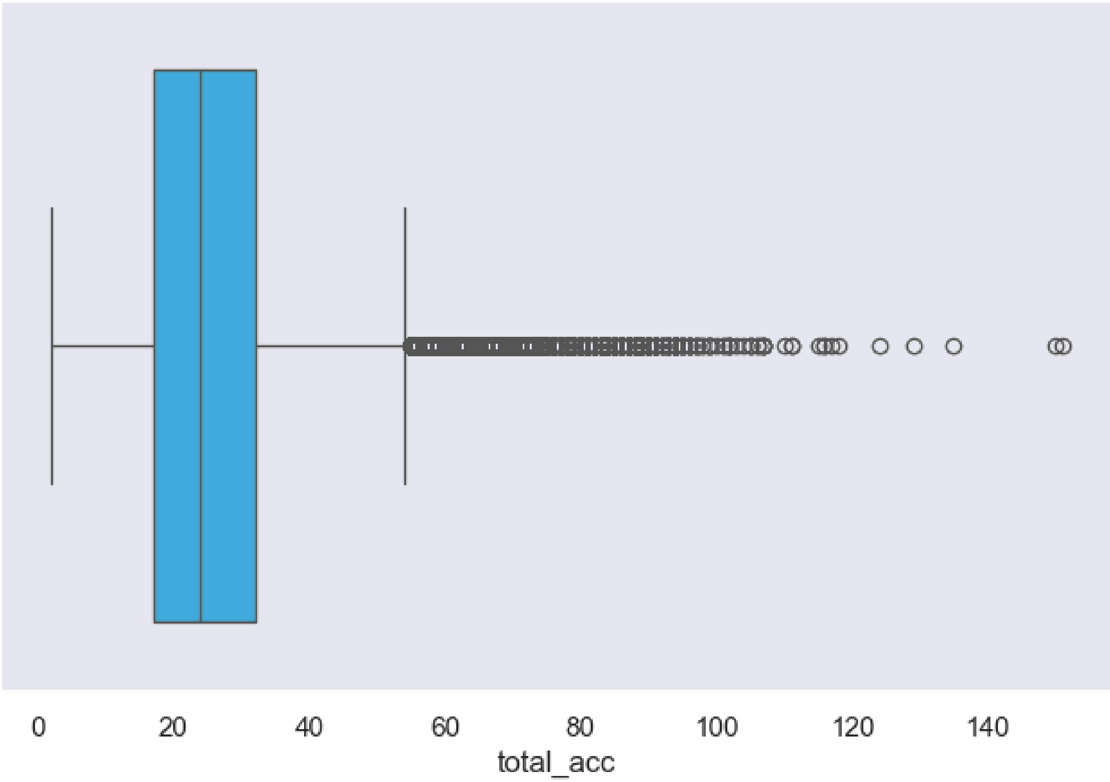
Boxplot for pub_rec



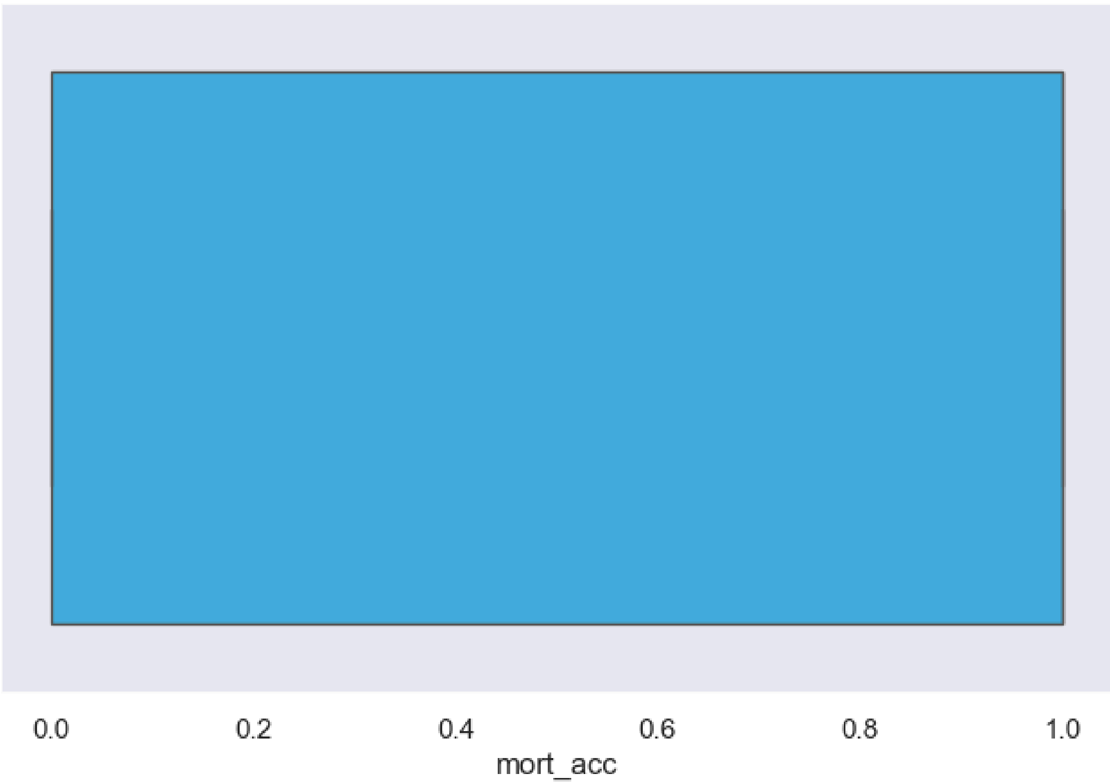

Boxplot for revol_bal
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
1e6
revol_bal

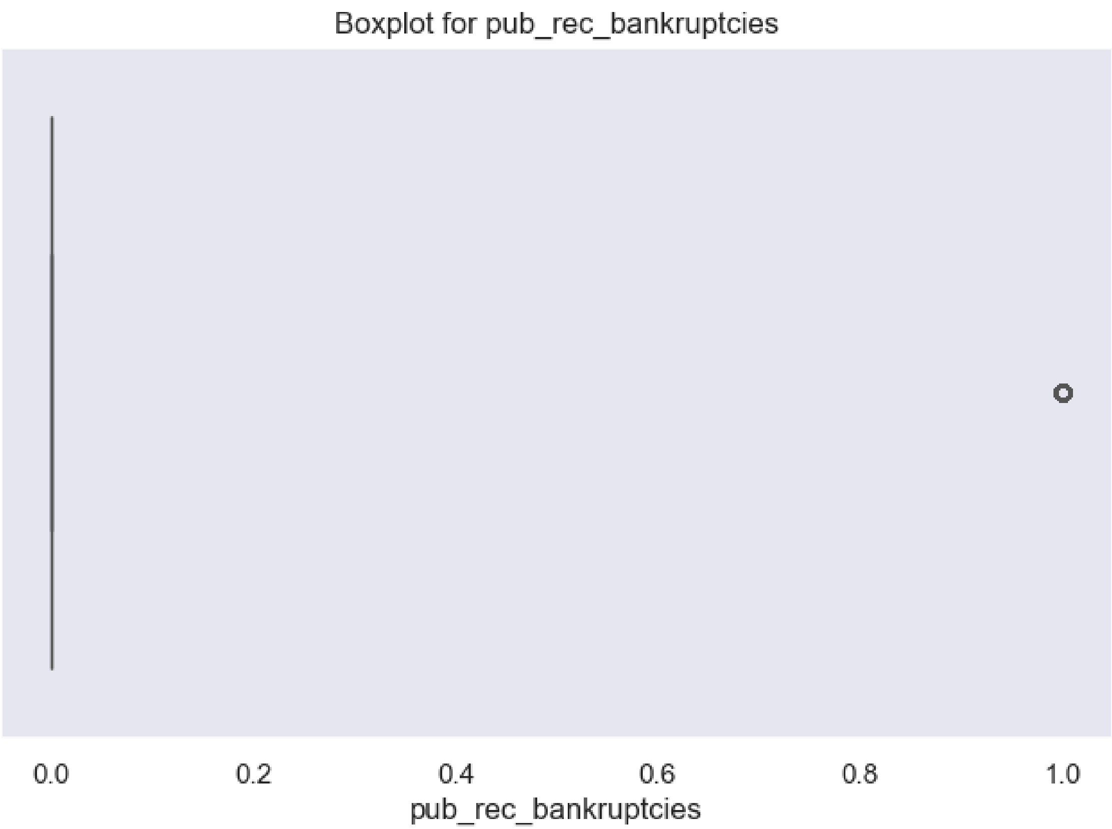

Boxplot for revol_util
0
200
400
600
800
revol_util

Boxplot for total_acc



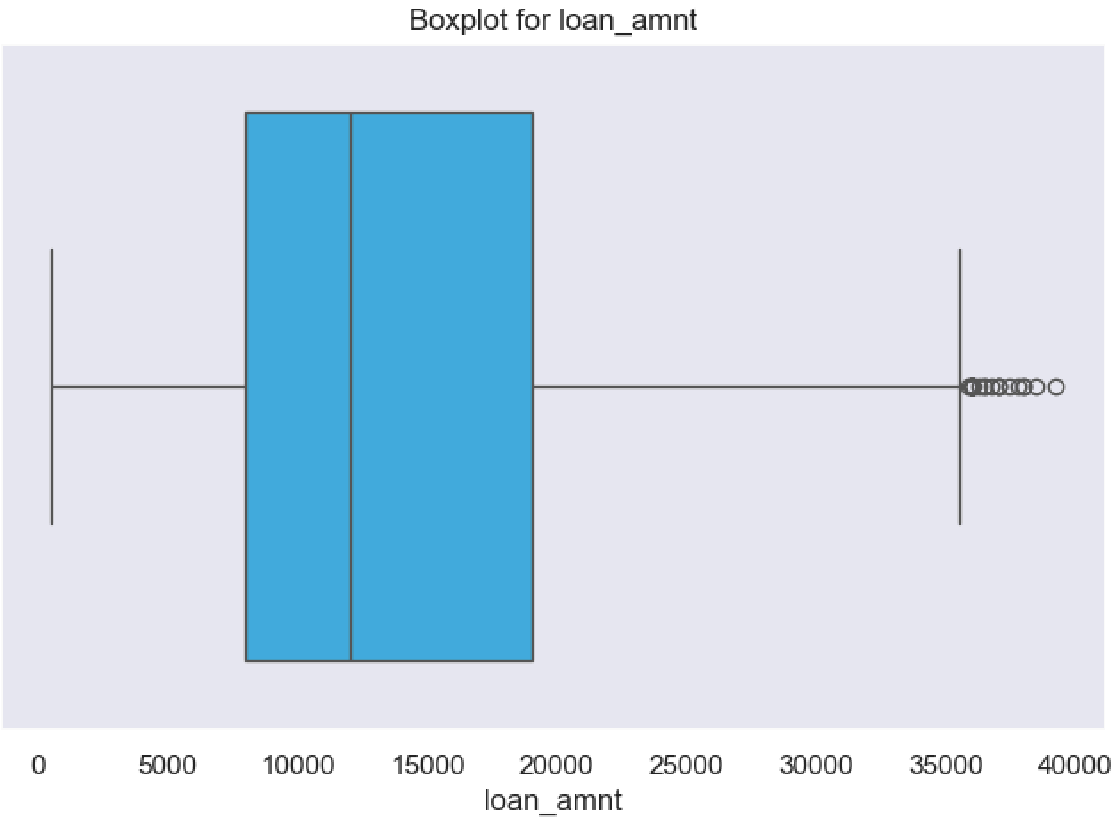
Boxplot for mort_acc

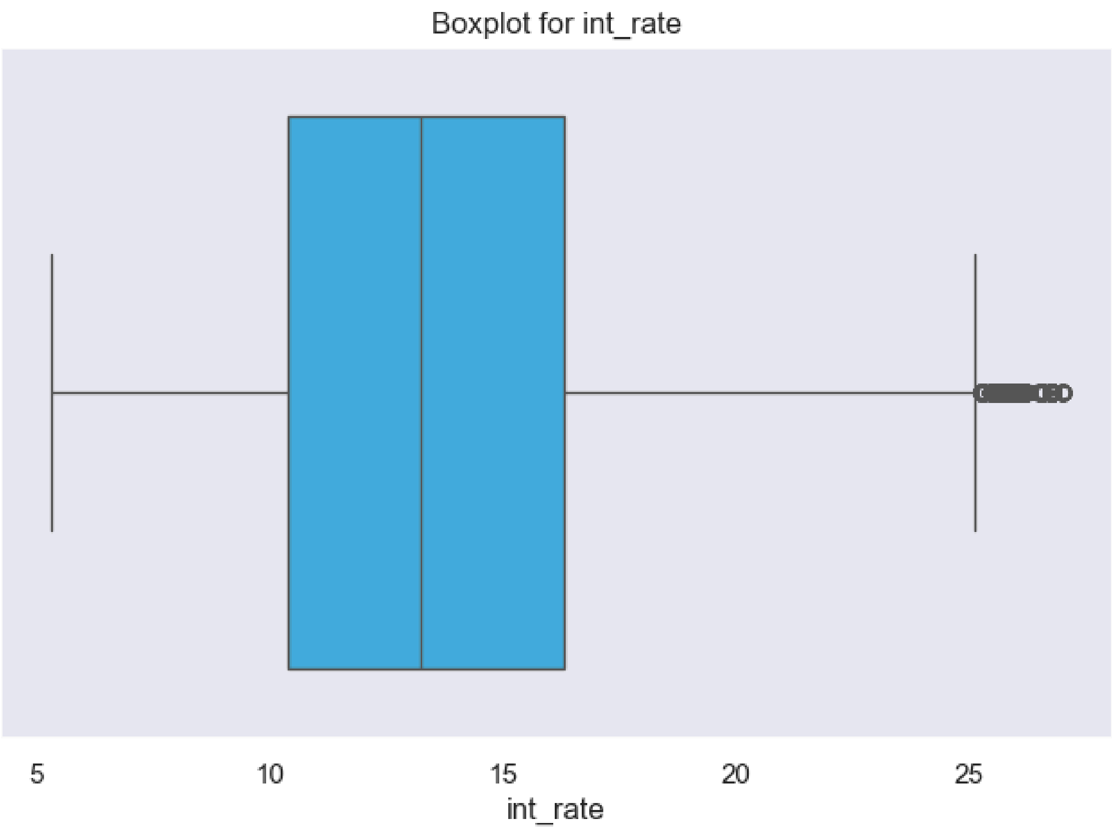


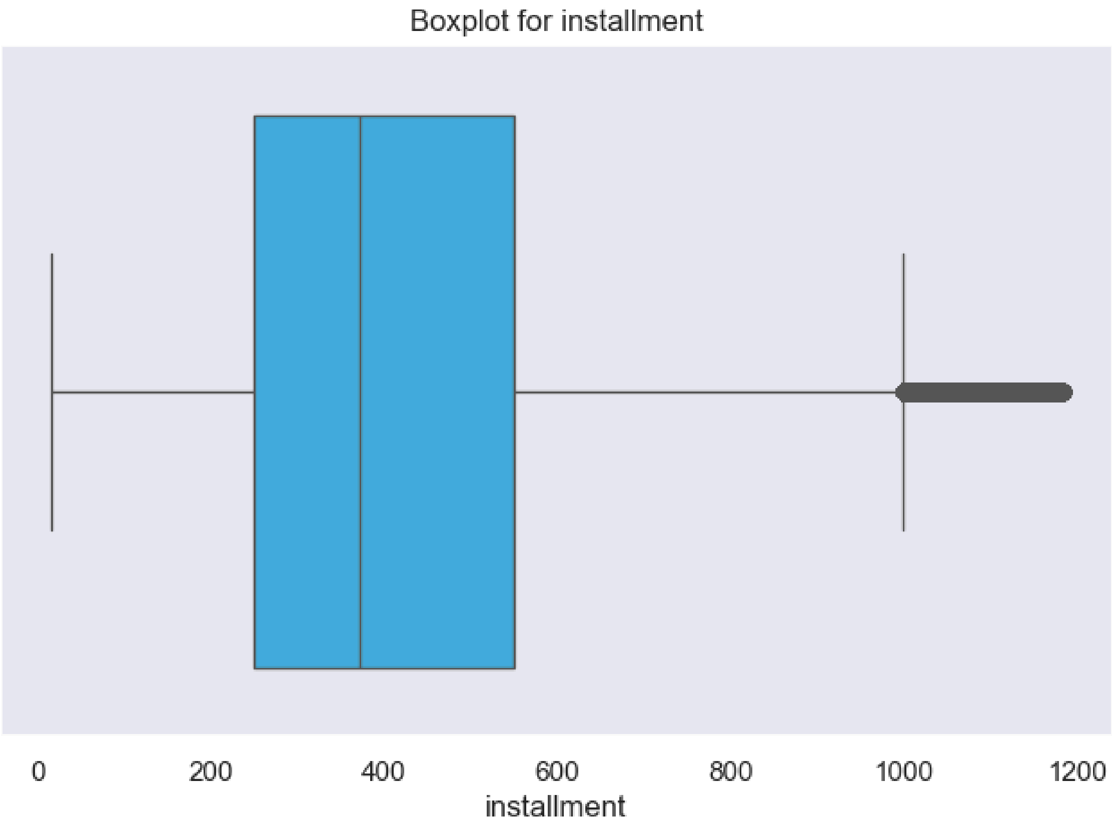


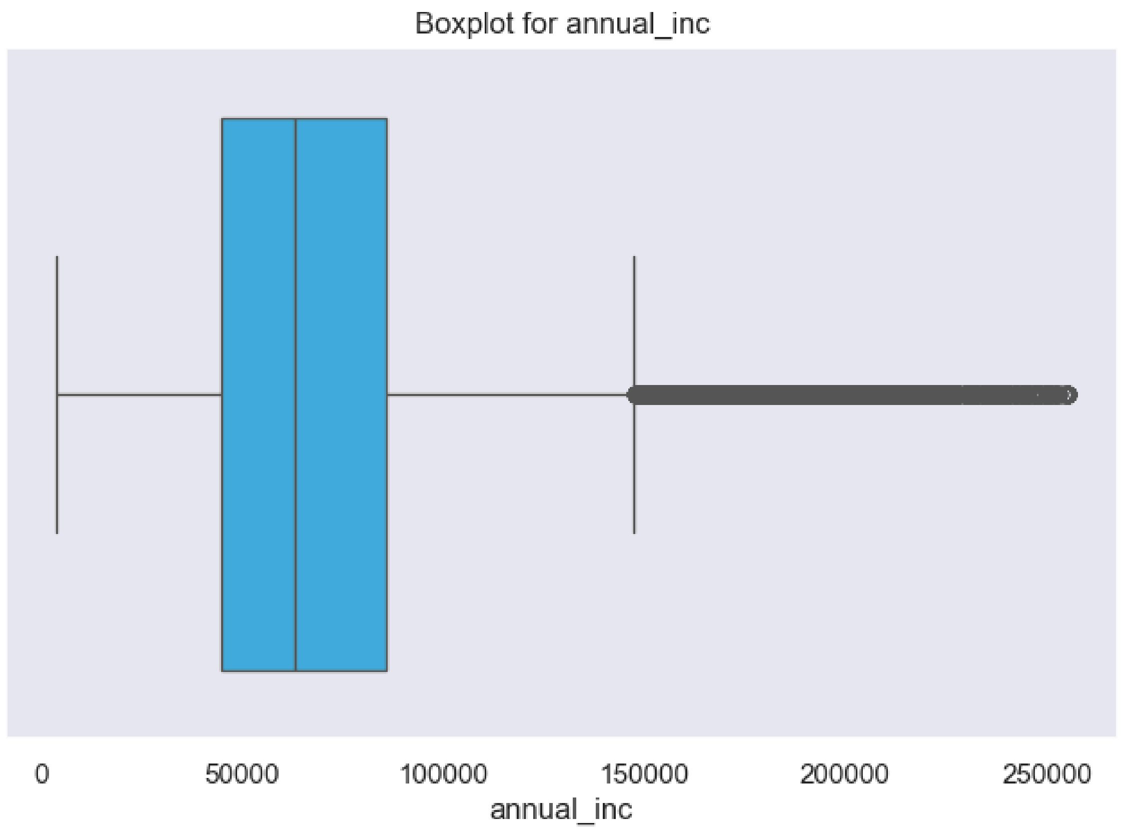
In [28]:
```
# Outlier treatment

for col in n_columns:
    if col in df.columns:
        mean = df[col].mean()
        std = df[col].std()
        upper_limit = mean + 3 * std
        lower_limit = mean - 3 * std
        df = df[(df[col] < upper_limit) & (df[col] > lower_limit)]
```

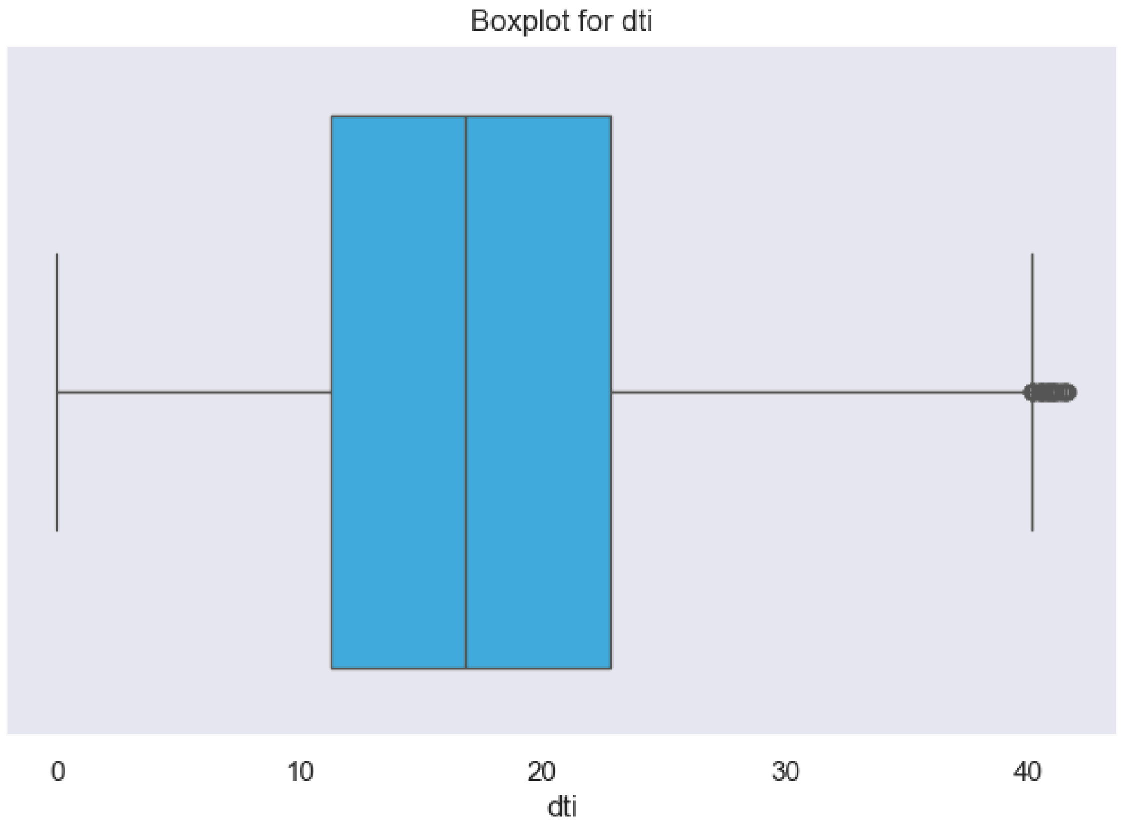
In [29]:
```
def box_plot(col):
    if col in df.columns:
        plt.figure(figsize=(8, 5))
        sns.boxplot(x=df[col],color="#29B6F6")
        plt.title('Boxplot for {}'.format(col))
        plt.show()
    else:
        print(f"Column '{col}' not found in the DataFrame.")

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

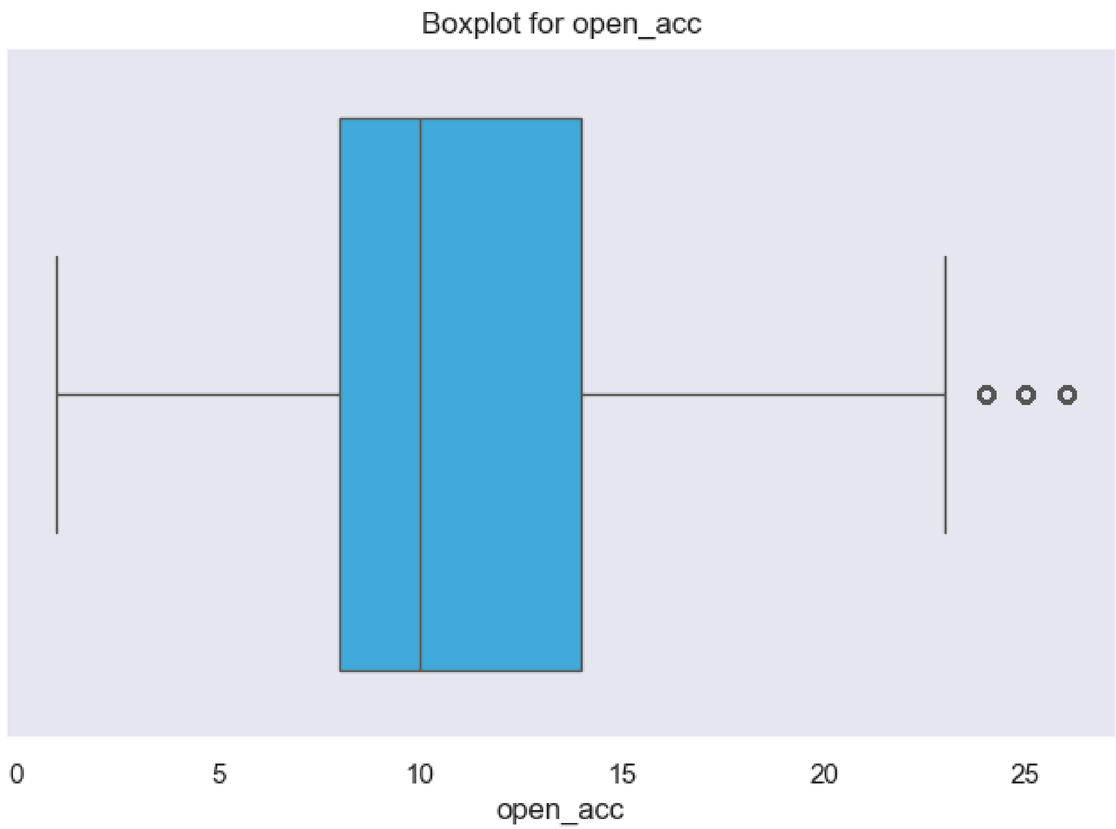


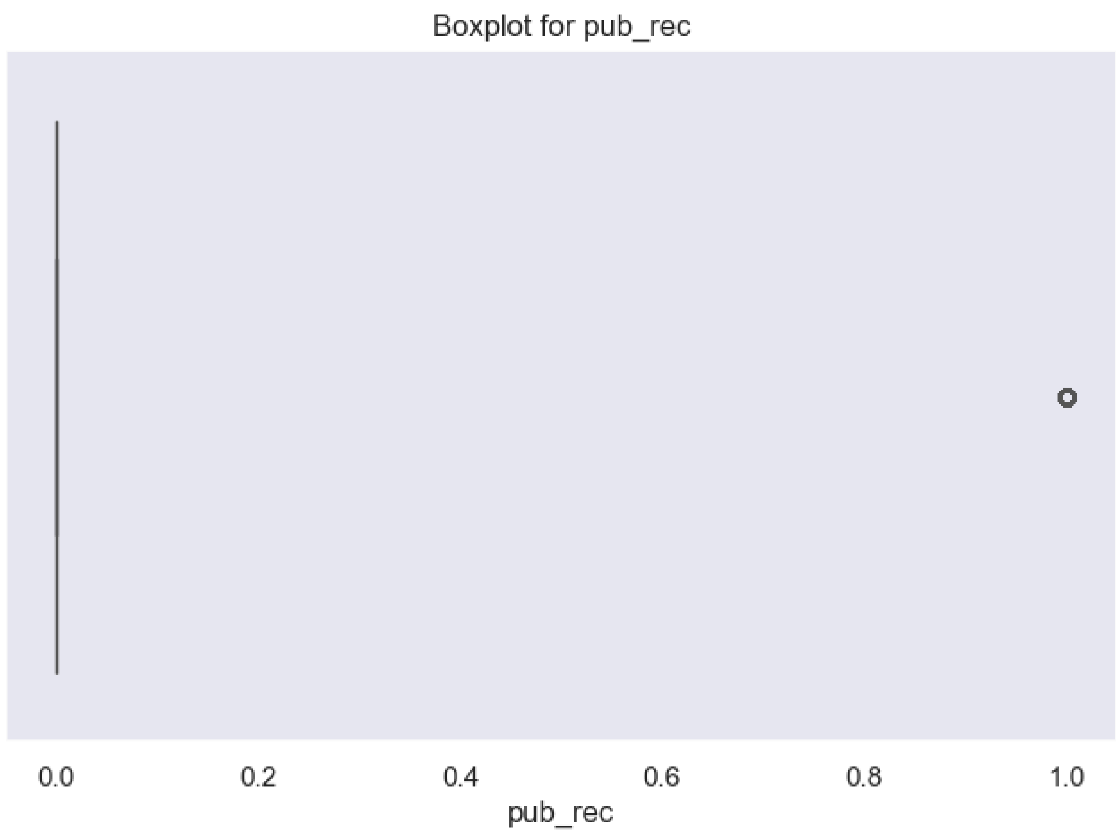

Boxplot for int_rate
5
10
15
20
25
int_rate

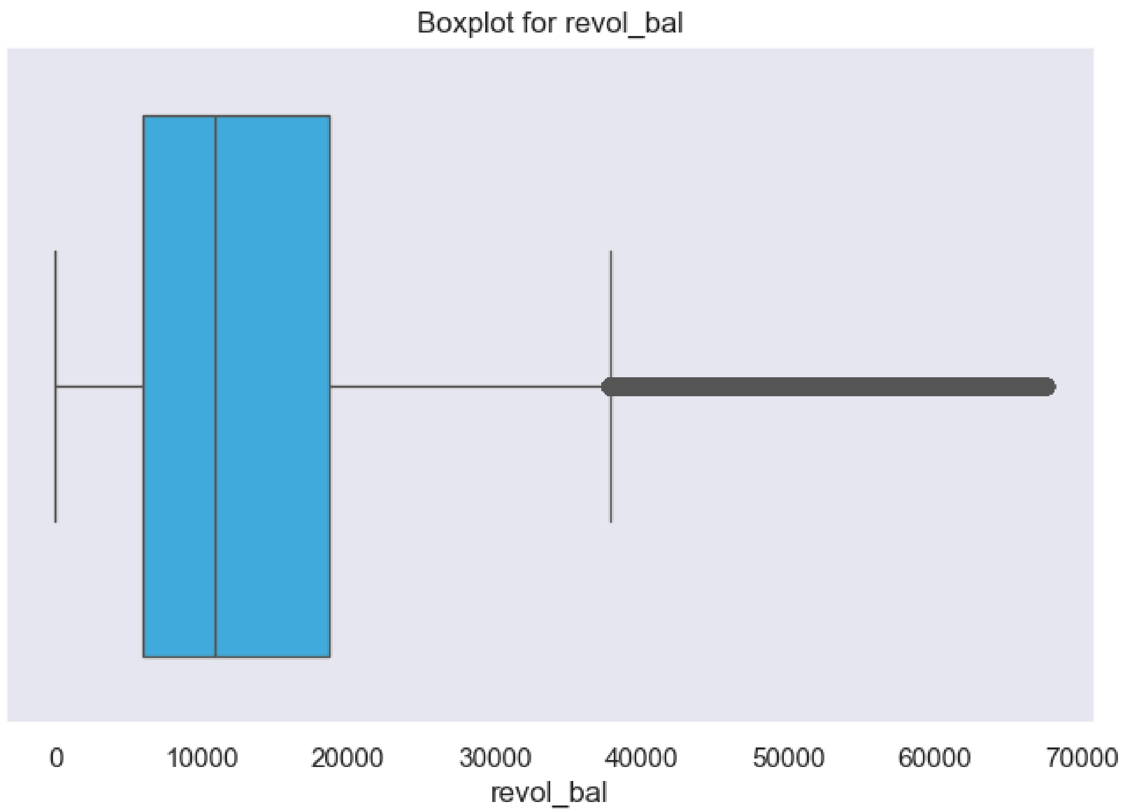

Boxplot for installment
0
200
400
600
800
1000
1200
installment

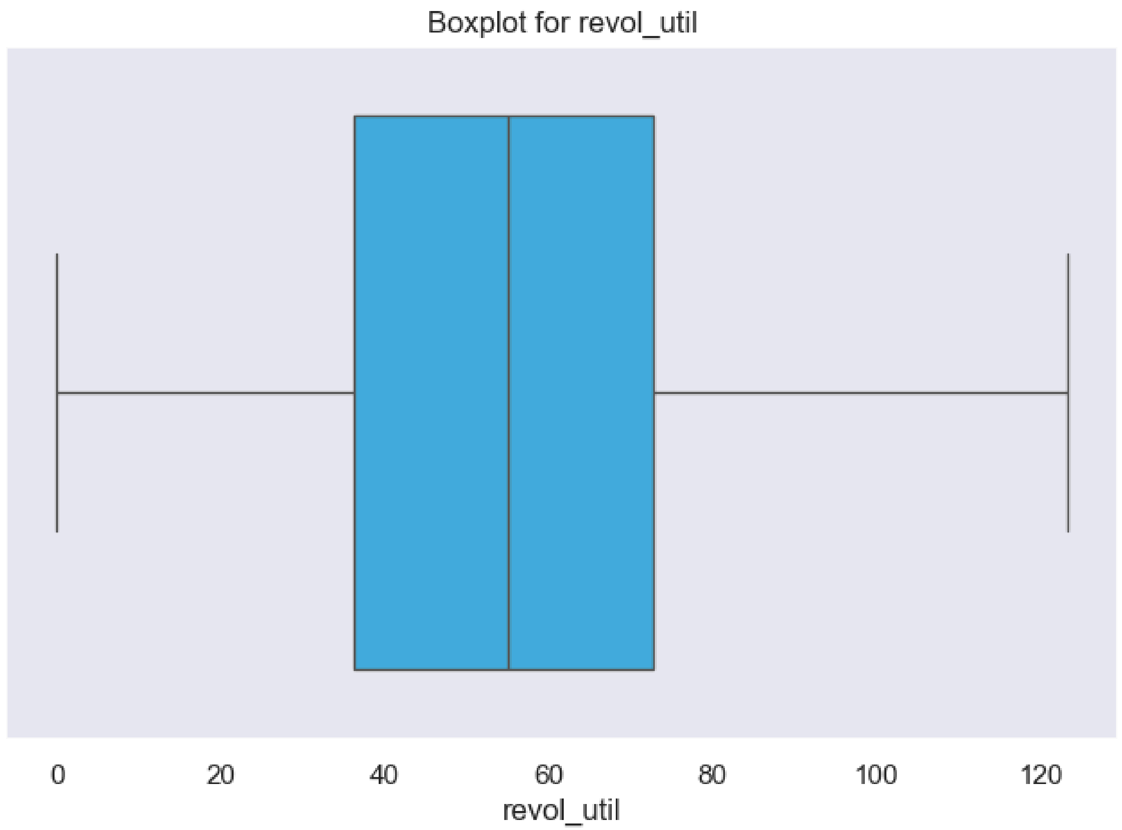

Boxplot for annual_inc
0
50000
100000
150000
200000
250000
annual_inc

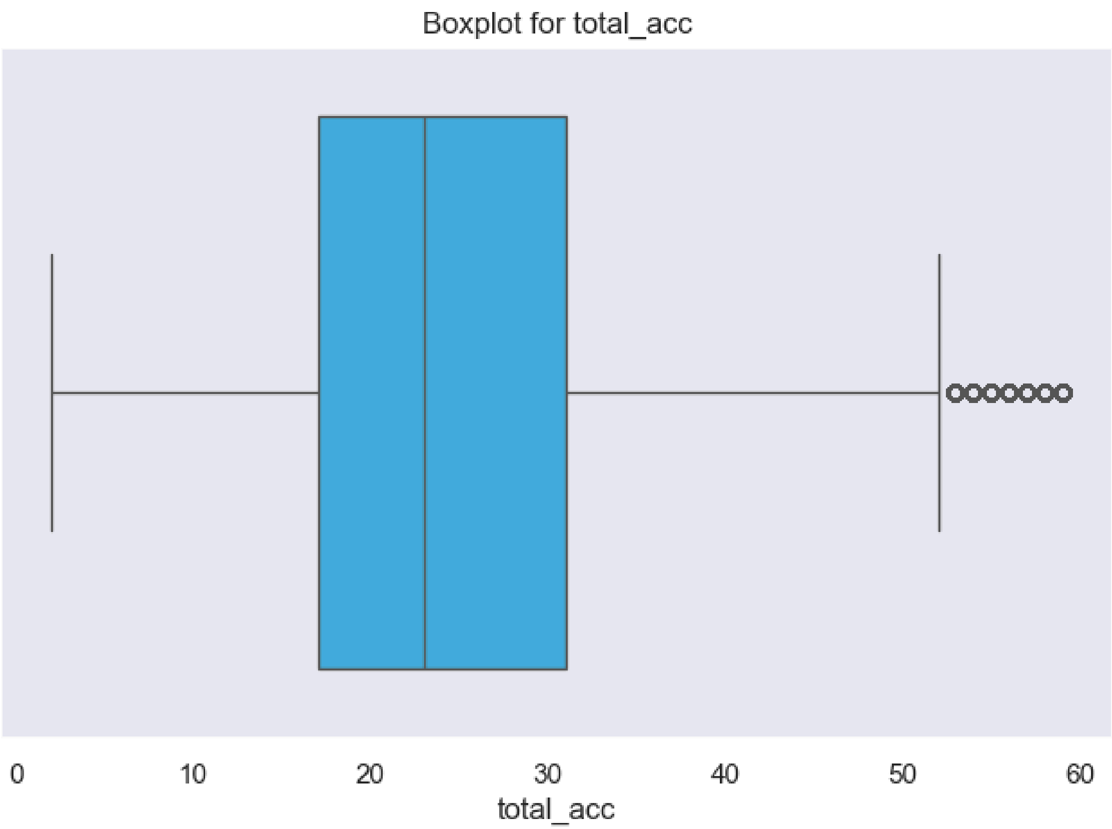

Boxplot for dti
0
10
20
30
40
dti

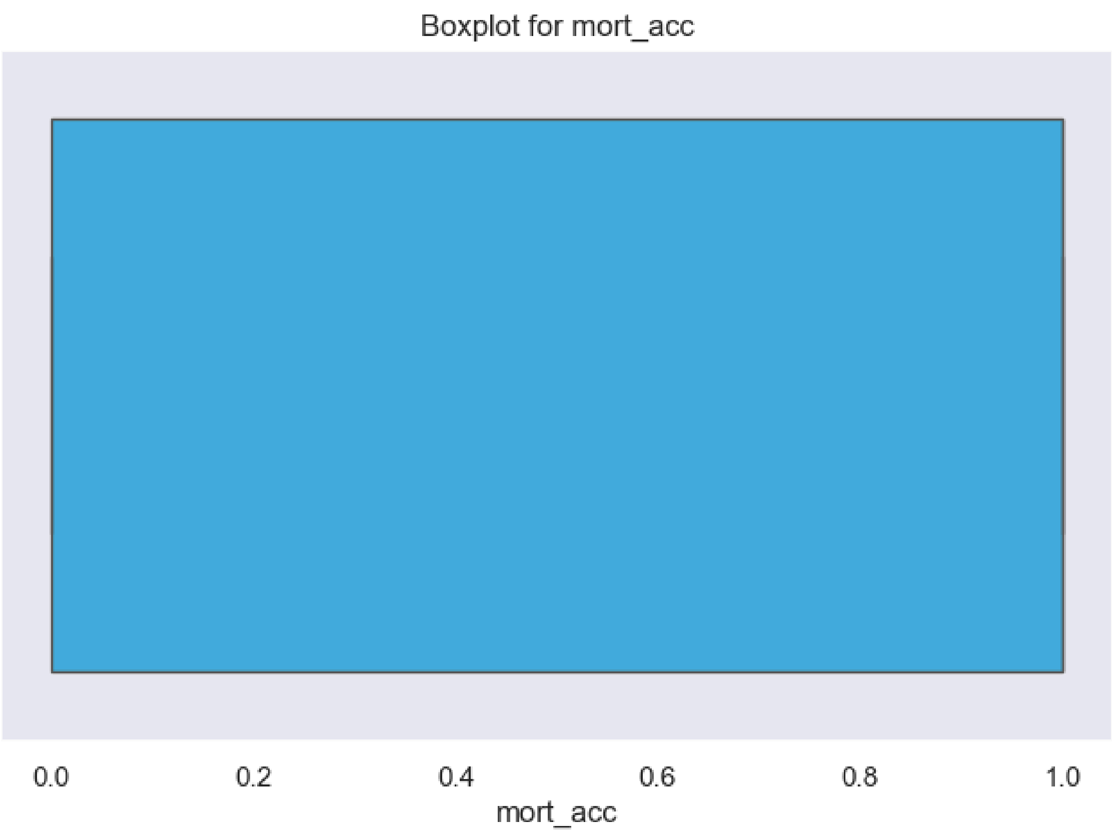

Boxplot for open_acc
0
5
10
15
20
25
open_acc

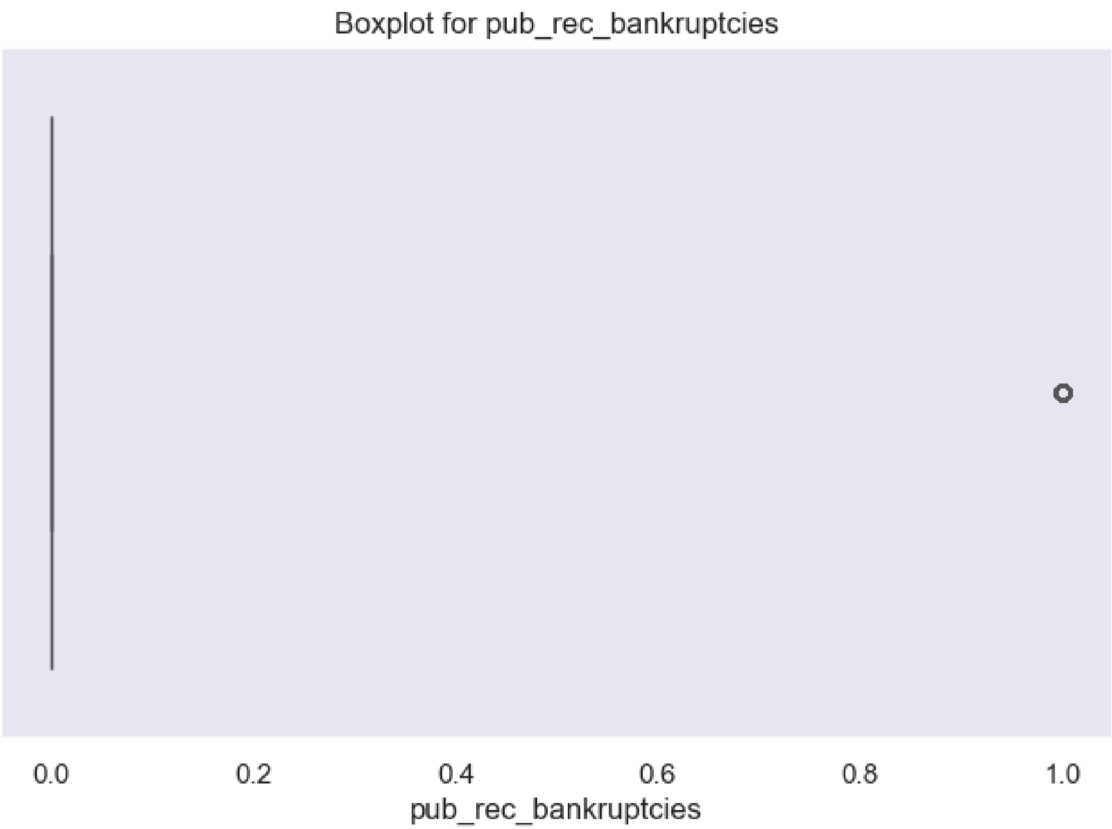

Boxplot for pub_rec
0.0
0.2
0.4
0.6
0.8
1.0
pub_rec


Boxplot for revol_bal
0
10000
20000
30000
40000
50000
60000
70000
revol_bal


Boxplot for revol_util
0
20
40
60
80
100
120
revol_util


Boxplot for total_acc
0
10
20
30
40
50
60
total_acc


Boxplot for mort_acc
0.0
0.2
0.4
0.6
0.8
1.0
mort_acc



In [30]:
```python
term_values = {' 36 months': 36, ' 60 months': 60}

df['term'] = df['term'].map(term_values)

df['loan_status'] = df['loan_status'].map({'Fully Paid': 0, 'Charged Off':

list_status = {'w': 0, 'f': 1}

df['initial_list_status'] = df['initial_list_status'].map(list_status)

df['zip_code'] = df['address'].apply(lambda x: x[-5:])

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

Out[30]:
```
zip_code
70466    14.375337
30723    14.289710
22690    14.272299
48052    14.127019
00813    11.605591
29597    11.548792
05113    11.519108
93700     2.768605
11650     2.762896
86630     2.730643
Name: proportion, dtype: float64
```

In [31]:
```
# Dropping of unnecessary columns

unnecessary_columns=['issue_d', 'emp_title', 'title', 'sub_grade','address',

df.drop(unnecessary_columns,axis=1, inplace=True)
```

One hot encoding

In [32]:
```
dummies=['purpose', 'zip_code', 'grade', 'verification_status', 'applicatio

data=pd.get_dummies(df,columns=dummies,drop_first=True)

pd.set_option('display.max_columns',None)

pd.set_option('display.max_rows',None)
```

In [33]:
```
from sklearn.model_selection import train_test_split

X=data.drop('loan_status',axis=1)
y=data['loan_status']
X_train, X_test, y_train, y_test =train_test_split(X,y,test_size=0.30,strat
print(X_train.shape)
print(X_test.shape)
```

```
(245249, 51)
(105108, 51)
```

Model Building

In [34]:
```
# 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_scor
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
```

In [35]:
```
scaler = MinMaxScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [36]:
```python
logreg=LogisticRegression(max_iter=1000)

logreg.fit(X_train,y_train)
```

Out[36]:

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

(https://scikit-learn.org/1.4/modules/generated/sklearn.linear_model.

In [37]:
```python
y_pred = logreg.predict(X_test)

print('Accuracy of Logistic Regression Classifier on test set: {:.3f}'.form
```

```
Accuracy of Logistic Regression Classifier on test set: 0.891
```

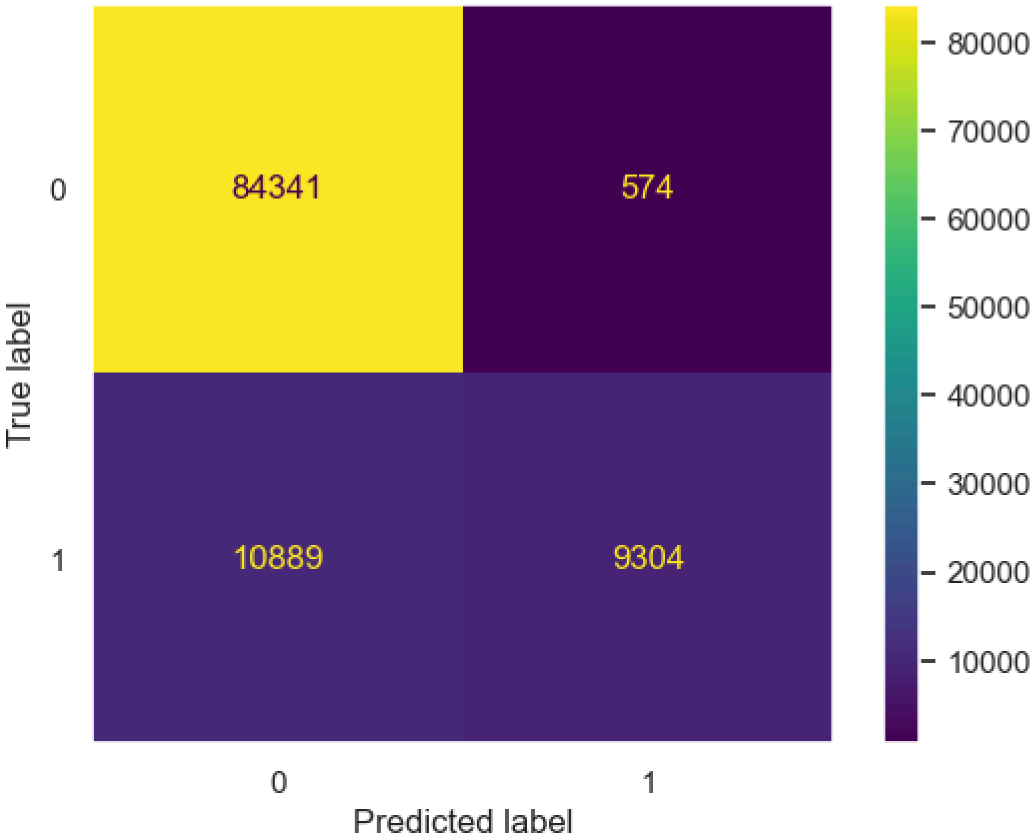
In [38]:
```python
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.89      0.99      0.94     84915
           1       0.94      0.46      0.62     20193

    accuracy                           0.89    105108
   macro avg       0.91      0.73      0.78    105108
weighted avg       0.90      0.89      0.88    105108
```

In [39]:
```
#Plot confusion Matrix

confusion_matrix=confusion_matrix(y_test,y_pred)

print(confusion_matrix)

ConfusionMatrixDisplay(confusion_matrix=confusion_matrix, display_labels=lo
```

```
[[84341   574]
 [10889  9304]]
```

Out[39]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x26a478
1cb10>
```



ROC Curve -

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

- True Positive Rate
- False Positive Rate

True Positive Rate (TPR) is a synonym for recall and is therefore defined as follows:

- TPR=(TP)/(TP+FN)

False Positive Rate (FPR) is defined as follows:

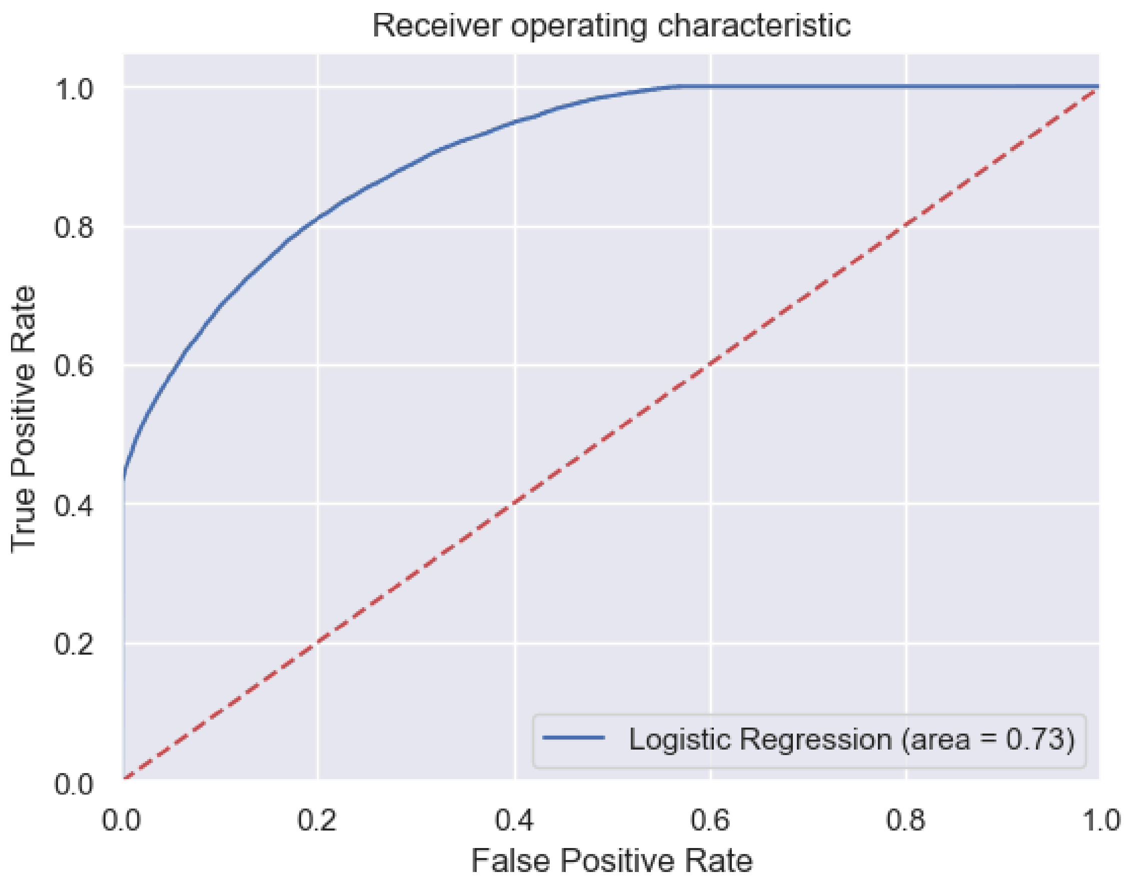
- FPR=(FP)/(FP+TN)

AUC (Area under the ROC Curve) -

AUC stands for "Area under the ROC Curve." That is, AUC measures the entire two-dimensional area underneath the entire ROC curve (think integral calculus) from (0,0) to (1,1).

AUC provides an aggregate measure of performance across all possible classification thresholds. One way of interpreting AUC is as the probability that the model ranks a random positive example more highly than a random negative example. For example, given the following examples, which are arranged from left to right in ascending order of logistic regression predictions:

In [40]:
```
logit_roc_auc = roc_auc_score(y_test, logreg.predict(X_test))

fpr, tpr, thresholds = roc_curve(y_test, logreg.predict_proba(X_test)[:,1])


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

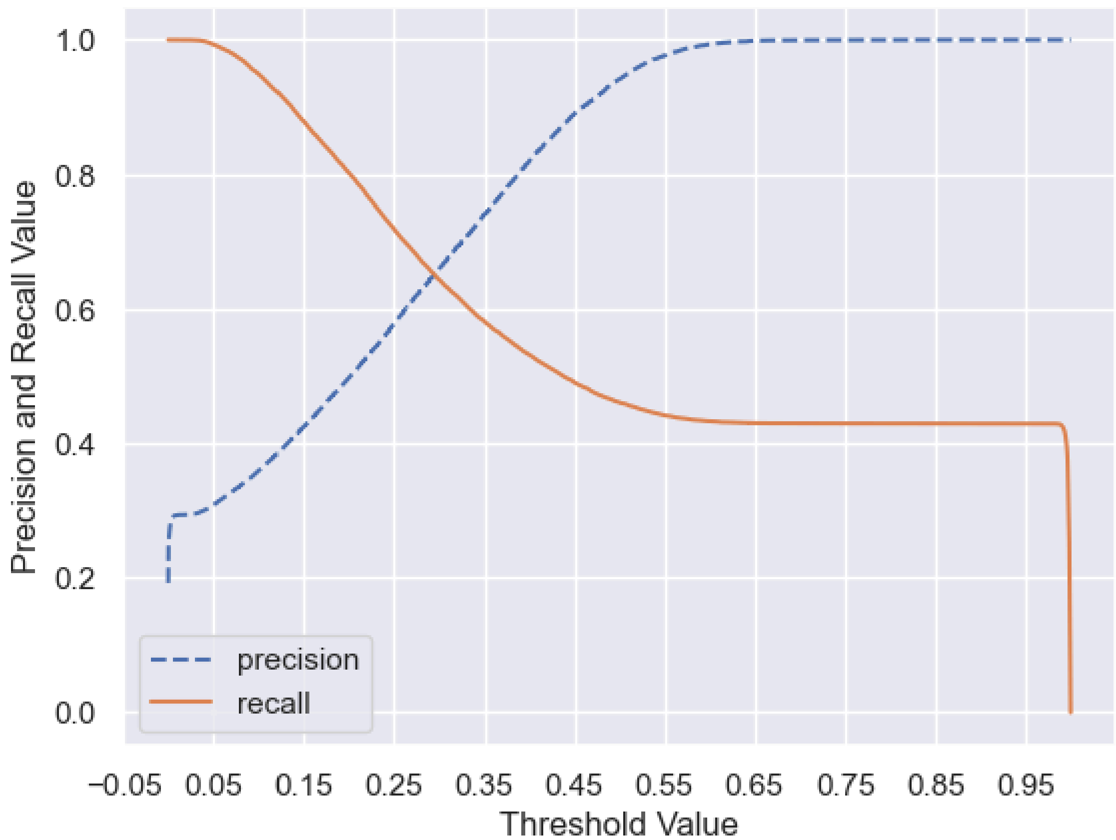


Insights:

- ROC-AUC curve is grossing the area near about 0.73 which indicates that model is performing well.
- There is still room for some model improvement
- By collecting more data, using a more complex model, or tuning the hyperparameters, it is possible to improve the model's performance.

Precision-Recall Curve

In [41]:
```
precisions, recalls, thresholds = precision_recall_curve(y_test, logreg.pre

threshold_boundary = thresholds.shape[0]

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

# 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()
```



Insights

- Precision score is highest at 0.55 threshold. High precision value indicates that model is positevly predicating the charged off loan status which helps business to take more stable decision.
- Recall score is higher on smaller threshold but after 0.55 the recall value is constant. Model is correctly classifying the actual predicated values as instances.

Actional Insights and Recommendations

1. 80% of the customers have paid the loan fully.
2. 20% of the customers are the defaulters.
3. The organization can the trained model to make prediction for whether a person will likely to pay the loan amount or he will be a defaulter.
4. Model achieves the 94% f1-score for the negative class (Fully Paid).
5. Model achieves the 62% f1-score for the positive class (Charged off).
6. Cross Validation accuracy and testing accuracy is almost same which infers model is performing the decent job. We can trust this model for unseen data
7. By collecting more data, using a more complex model, or tuning the hyperparameters, it is possible to improve the model's performance.
8. ROC AUC curve area of 0.73, the model is correctly classifying about 73% of the instances. This is a good performance, but there is still room for improvement.
9. The precision-recall curve allows us to see how the precision and recall trade-off as we vary the threshold. A higher threshold will result in higher precision, but lower recall, and vice versa. The ideal point on the curve is the one that best meets the needs of the specific application.
10. After balancing the dataset, there is significant change observed in the precion and recall score for both of the classes.
11. Accuracy of Logistic Regression Classifier on test set: 0.891 which is decent and not by chance